НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК _____

«До захисту допущено»

Науковий керівник
кафедри

_____ І.А. Дичка

«___»_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Метод растеризації зрізів тривимірних медичних зображень»

Виконав:
студент VI курсу, групи КП-61м
Черних Денис Андрійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Сулема Є. С. _____

Консультант з _____:
Ст. викладач кафедри ПЗКС, к.т.н.,
Люшенко Л.А. _____

Рецензент:
В.о. зав. кафедри ММСА ІПСА, к.т.н., доцент,
Тимощук О.Л. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2018 року

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА СКОРОЧЕНЬ

Алгоритм Брезенхема для побудови ліній – алгоритм для растеризації відрізка прямої між двома заданими точками, що використовує лише цілочисельну арифметику;

Воксель – частина дискретного простору, яка є найменшою складовою воксельної моделі; є аналогом пікселя у тривимірному просторі;

Воксельна модель – тривимірне зображення об’єкту, яке побудоване із вокселів;

Медична візуалізація – процес візуального представлення внутрішньої будови органів та тканин тіла з метою отримання інформації для подальшого аналізу або ж медичного втручання;

Растеризація – процес побудови растрового зображення;

Рендеринг – процес візуалізації віртуальних об’єктів тривимірної сцени (моделей) за допомогою спеціальних програмних засобів;

Ткацькі алгоритми растеризації – сімейство алгоритмів для растеризації площин у тривимірному просторі, в яких растеризація заданої площини відбувається шляхом копіювання растеризованої master-лінії уздовж base-лінії;

.NET (.NET Framework) – програмна платформа, розроблена компанією Microsoft, для розробки програмного забезпечення та web-застосунків;

ANALYZE – пакет програмного забезпечення, а також файловий формат для збереження, багатовимірного відображення та обробки тривимірних медичних зображень;

Base-лінія – це растеризований відрізок прямої, який задає напрямок руху master-лінії при растеризації площини ткацьким способом;

CLR – Common Language Runtime – це компонент пакету Microsoft .NET Framework, віртуальна машина, на якій виконуються всі мови платформи .NET Framework;

DDA – Digital Differential Analyzer – алгоритм для растеризації відрізка прямої між двома заданими точками;

DICOM – Digital Imaging and Communications in Medicine – стандарт яким визначаються процеси збереження, створення, передачі та візуалізації медичних зображень; також однойменний файловий формат;

Master-лінія – це растеризований відрізок прямої, який копіюється уздовж base-лінії паралельним перенесенням та утворює поверхню площини;

NIFTI – Neuroimaging Informatics Technology Initiative – файловий формат для збереження і візуалізації воксельних моделей даних;

STL – Standard Template Library – бібліотека узагальнених алгоритмів, контейнерів та функцій для використання у мові програмування C++.

ВСТУП

Дослідження зрізів воксельних моделей даних є розповсюдженою задачею у медичній галузі. Їх всебічне вивчення дозволяє фахівцям більше дізнатися про структуру досліджуваного об’єкту, а також отримати чітке уявлення про організацію його складових. Це має першочергове значення для діагностики та наукових експериментів.

Найчастіше виникає потреба дослідити саме внутрішню будову об’єкту, і, в такому разі, дослідників більше цікавлять окремі частини моделі, зокрема, зрізи воксельних даних. Зрізом є множина вокселів, що належать заданій площині зрізу, або ж максимально точно її відтворюють. Виходячи із специфіки воксельних моделей даних, спосіб одержання зрізів залежить від кута, під яким виконується цей зріз. Оскільки воксельна модель представляє собою тривимірний масив даних, то найбільш тривіально задача растеризації зрізу вирішується при кутах, кратних 90°. В загальному ж випадку, для растеризації зрізів під довільними кутами, необхідно використовувати деякі спеціальні методи, що дозволяють отримати потрібний зріз під довільним кутом.

Такі методи відрізняються між собою підходами, які використовуються для визначення відповідної множини вокселів, яка представляє собою площину зрізу. Хоча дані методи і призначені для вирішення однієї і тієї ж задачі, проте отримані результати растеризації зрізів можуть відрізнятися між собою. До зрізів воксельної моделі можуть висуватися різні вимоги, такі як точність зрізу, товщина зрізу, ступінь зв’язності вокселів та інші. Відповідно, вибір методу для растеризації необхідно здійснювати залежно від конкретного випадку.

В рамках даного дослідження були розглянуті методи та алгоритми растеризації площин та полігонів у тривимірному просторі. В результаті проведеного аналізу існуючих рішень, було запропоновано власний метод растеризації зрізів тривимірних медичних зображень.

Запропонований метод растеризації дозволяє отримувати зрізи воксельних моделей даних під довільними кутами. Отримані зрізи мають одиничну товщину, та достатньо точно відтворюють площину зрізу. Розроблений метод растеризації використовує лише цілочисельну арифметику та мінімізує кількість обчислень в циклах растеризації фрагментів master-лінії, що робить використання методу достатньо ефективним для растеризації зрізів воксельних моделей.

Дослідження виконане у рамках проекту "EU-Ukrainian Mathematicians for Life Sciences (EUMLS)" європейської наукової програми FP7.

1. АНАЛІЗ МЕТОДІВ ВІЗУАЛІЗАЦІЇ ТРИВИМІРНИХ МОДЕЛЕЙ

1.1. Загальна характеристика воксельних моделей

Головною метою, яка ставиться при створенні тривимірних зображень, є візуалізація певного набору даних, що стосуються предметної області. Зазвичай, такими наборами даних є параметри об’єктів реального світу, які представляють інтерес для конкретної галузі, для яких необхідно створити відповідні моделі. Тривимірні зображення дозволяють представити досліджувані об’єкти або сукупність їхніх характеристик як набори даних, які мають певні координати у тривимірному просторі. Найчастіше, це деякі фізичні характеристики об’єкту, що моделюється.

Зазвичай, створення тривимірних зображень знаходить своє застосування в таких галузях, у яких недостатньо побудувати полігональну модель об’єкту. Полігональна модель добре підходить для представлення форми об’єкту, що моделюється, та його візуалізації, проте така модель не містить жодної інформації про внутрішню структуру об’єкту, його внутрішню будову або інші дані всередині. Тому виникає необхідність у створенні тривимірних зображень. Також тривимірні зображення добре підходять для моделювання неперервних середовищ та полів значень.

В деяких випадках використання полігональних моделей може значно ускладнити реалізацію програмного забезпечення для візуалізації графічних сцен, особливо, якщо необхідно відобразити складні графічні ефекти, на кшталт, хвиль на поверхні води. Інколи, відображення певного набору даних є взагалі неможливим із використанням полігональних моделей, і в цьому випадку, єдиним допустимим рішенням буде застосування вокселізації цих даних [1, 2].

Воксельне моделювання може використовуватись у різноманітних областях людської діяльності, тобто всюди, де необхідно отримати модель, яка надає адекватне представлення внутрішнього стану об’єкту. Так, у промисловості використовуються спеціалізовані промислові томографи,

які надають можливість отримати тривимірні зображення складних деталей та механізмів, що дозвляє отримати чітке представлення про взаємодію внутрішніх складових досліджуваного об’єкту [3].

У різноманітних засобах для моделювання, а також в ігровій індустрії, широко використовуються техніки та прийоми для побудови воксельних моделей ландшафтів. У комп’ютерній графіці такі воксельні ландшафти використовуються замість карт висот, тому що вони надають можливість представити різноманітні нерівності рельєфу, такі як пагорби, виступи, печери, пустоти та ін. Не всі перелічені особливості ландшафту можуть бути представлені на відповідних картах висот через те, що такі карти задають лише висоту поверхні, але не містять інформації про структуру нижче або вище заданого рівня [4].

Проте найчастіше тривимірні зображення знаходять своє застосування у галузі медицини, оскільки там, зазвичай, основний інтерес викликає внутрішній стан об’єкту, а не його зовнішній вигляд. Також варто зазначити, що у більшості випадків буває неможливо дослідити об’єкт із середини через його недоступність або без ризику завдати йому шкоди.

У ході обстеження пацієнтів медичних закладів, проведення різноманітних лабораторних досліджень та експериментів, у процесі діагностики захворювань використовуються сучасні методи діагностики, такі як магнітно-резонансна томографія, комп’ютерна томографія, ультразвукова діагностика тощо. Дані технології дозволяють отримати тривимірні медичні зображення без жодних руйнуючих наслідків для об’єкту дослідження. Такі зображення надають медичним працівникам можливість побачити та досконало вивчити внутрішній стан досліджуваного об’єкту і зробити правильні висновки.

За аналогією до двовимірних зображень, які можна представити як двовимірну матрицю, що складається із елементів – пікселів, – тривимірні

зображення можна представити як тривимірний масив даних, де елементарною одиницею є воксель.

Воксель – певна частина тривимірного простору, яка має власні координати у цьому просторі і є елементом тривимірного зображення. Координати вокселя є відносними та визначаються відносно інших, сусідніх, вокселів. Вокселі мають певне значення, певну величину, яка є характеристикою модельованого об’єкта у даній точці. Наприклад, воксель може представляти собою значення прозорості або густини фізичного тіла у певній точці або одиниці об’єму тіла. Сам по собі, воксель може бути представлений у просторі як певна тривимірна фігура. Найчастіше, вокселі подаються у формі куба. Розмір такого куба визначається роздільною здатністю пристрою, яким виконувалося сканування об’єкту тривимірного зображення. Відповідно, тривимірне зображення є воксельною моделлю об’єкта, що досліджується.

Воксельна модель – це певна множина вокселів, що розглядаються в одному тривимірному дискретному просторі. В такому дискретному просторі координати точок, зазвичай, вважаються цілочисельними. Фізично, відстань між двома сусідніми точками у такому просторі може задаватися по різному для різних напрямків, паралельних до координатних осей, але найчастіше усі відстані між двома сусідніми точками в усіх напрямках є однаковими [5].

В загальному випадку, воксельна модель може складатися із вокселів, які можуть бути розташовані довільним чином у такому просторі. Проте, через специфіку побудови воксельних моделей, особливо, тривимірних медичних зображень, де такі моделі будуються шляхом поєднання послідовностей двовимірних відсканованих зображень, вокселі у моделях розташовуються послідовно та неперервно. Для збереження значень вокселів, зазвичай використовується тривимірних масив даних, де доступ до кожного значення, тобто, вокселя, здійснюється за відповідними індексами по трьох вимірах.

В тривимірних медичних зображеннях може бути додано та використано четверту розмірність – вимір часу. В такому випадку, отримана воксельна модель може використовуватись для представлення зміни внутрішнього стану об’єкту із плином часу, що особливо актуально при діагностиці певних захворювань, зокрема, серцево-судинної системи.

Для воксельних моделей даних характерна певна специфіка. Як було зазначено вище, дані, які представлені у таких моделях можна уявляти як тривимірний масив. Самі по собі, вокселі не містять жодної інформації про своє місцезнаходження у просторі. Їхні координати визначаються на основі їх позиції у такому масиві та роздільної здатності пристрою, яким було отримано дані моделі. Такі моделі у не стиснутому вигляді здатні займати великі об’єми пам’яті. Для порівняння, полігональна модель містить у собі вершини моделі, які мають вже обчислені координати у тривимірному просторі. Для одного й того ж самого об’єкту, така модель може забезпечити більш якісну візуалізацію і при цьому займає набагато менше пам’яті. Проте інформація про внутрішнє подання досліджуваного об’єкту буде втрачена, оскільки полігональна модель всередині є порожньою.

Воксельні моделі можуть бути оптимізовані. Дані можуть бути подані у вигляді розрідженого воксельного октодерева [6, 7]. Концепція, яка покладена в основу такої структури даних полягає в розбитті простору на менші підпростори. Відповідно до даної концепції, перший вузол такого дерева – корінь, представляє собою куб, який містить весь об’єкт уцілому. Будь-який із вузлів може мати вісім нащадків, або не мати жодного нащадка. Інакше кажучи, кожен куб – це простір, який відповідає вузлу дерева, та розбивається на вісім октантів – менші підпростори – які є дочірніми вузлами для батьківського вузла.

Дана технологія дозволяє якісно візуалізувати воксельну модель і при цьому оптимізувати витрати пам’яті та швидкодію. Проте в деяких особливих випадках, використання такої структури даних для зменшення кількості вокселів усередині об’єкту є небажаним, оскільки інформація про

внутрішню структуру об’єкту може бути втраченою. Тому тривимірні медичні зображення, найчастіше, зберігають не стиснутими.

1.2. Методи побудови тривимірних медичних зображень

Медична візуалізація – це техніка та процес створення візуального представлення про внутрішність тіла для клінічного аналізу та, можливо, подальшого медичного втручання, а також візуальне зображення функціонування деяких органів або тканин [8, 9]. Медична візуалізація спрямована на виявлення та подання внутрішніх структур, прихованих за шкірою та кістками, а також для діагностики та лікування захворювань. Медична візуалізація надає змогу зібрати дані про нормальну анатомію і фізіологію живих організмів, що дозволяє виявити аномалії в подальших дослідженнях. Хоча візуалізація окремо взятих органів та тканин також може бути виконано для досягнення медичних цілей, такі процедури зазвичай не розглядаються як складова медичної візуалізації.

Вимірювання і запис тривимірних медичних зображень проводиться методами, які, загалом, не призначені для отримання звичайних зображень, такими як електроенцефалографія, магнітоенцефалографія, електрокардіографія та ін., і являє собою сукупність технологій, які надають дані, що подаються у вигляді функції часу або карти, яка містить інформацію про місця вимірювань.

Медична візуалізація часто сприймається як набір методів, які неінвазивним шляхом (без введення інструментів в організм пацієнта) створюють зображення внутрішнього аспекту тіла. У цьому вузькому сенсі, медичну візуалізацію можна розглядати як рішення математичних зворотніх задач. Це означає, що причина (властивості живої тканини) виводиться з ефекту (спостережуваного сигналу). Наприклад, у разі використання ультразвукової діагностики для проведення дослідження, такими ефектами будуть ультразвукові хвилі та луни, які будуть надходити від тканин тіла.

Багато різновидів медичного апаратного устаткування, серед яких сканери комп'ютерної томографії, тривимірної ультразвукової діагностики, магнітно-резонансної томографії тощо при виконанні сканування видають пошарову інформацію. Після завершення сканування вибудовується тривимірне зображення із вокселів. Їх значення відображають дані, отримані з пристрою. Наприклад, якщо мова йде про комп’ютерну томографію, то цими даними може бути прозорість для рентгенівських променів.

Отримані воксельні моделі даних, наприклад, після виконання магнітно-резонансної томографії, можуть бути розкладені на двовимірні зрізи даних та проаналізовані спеціалістами. Таким чином, стає можливим вивчення практично будь-якого зрізу інформації.

1.2.1. Комп'ютерна томографія

Комп’ютерна томографія – це один із методів сканування рентгенівськими променями [10]. Поступово проходячи через тканини тіла досліджуваного об’єкту у різних напрямках, рентгенівський промінь здійснює пошарове сканування. Частіше за все, під терміном комп’ютерна томографія розуміють рентгенівську комп’ютерну томографію. З її допомогою можна отримати інформацію про внутрішню будову тіла людини, розрізнити будь-який орган, визначити його розмір, розташування, форму, дізнатися про стан поверхні та загальну будову, його функції та густину. З використанням комп’ютерної томографії можна досліджувати будову та роботу кровообігу.

Метод заснований на вимірюванні та комп’ютерній обробці різниці послаблення рентгенівського випромінювання різними за густиною тканинами.

Під час застосування комп’ютерної томографії використовують, так звані, контрастні речовини, щоб можна було чітко розрізняти знімки на екрані. Сучасні комп’ютерні томографи надають можливість одержати надзвичайно тонкі зображення – від 0,5 до 10 мм. Отримані зображення

можуть бути двовимірними або ж тривимірними. Після їх створення можна виконувати їх подальшу обробку (поворот, масштабування тощо).

Для візуальної і кількісної оцінки щільності структур, що візуалізуються методом комп'ютерної, використовується шкала ослаблення рентгенівського випромінювання, що отримала назву шкали Хаунсфілда. Діапазон одиниць шкали, що відповідають ступеню ослаблення рентгенівського випромінювання анатомічними структурами організму, становить від -1024 до +3071, тобто 4096 значень. Середній показник у шкалі Хаунсфілда відповідає щільності води, негативні величини шкали відповідають повітрю і жировій тканині, позитивні – м'яким тканинам, кістковій тканині і більш щільним речовинам [11]. На практиці, виміряні показники ослаблення можуть дещо відрізнятися на різних апаратах.

1.2.2. Діагностика з допомогою ультразвуку

Дуже важливим засобом для проведення діагностування пацієнтів є ультразвукова діагностика, яка знайшла широке застосування у різних областях медицини. Основний принцип діагностики із допомогою ультразвуку полягає у генерації звукових хвиль спеціальними апаратними засобами (так званими, випромінювачами) [12].

Забезпечення найбільш ефективної передачі енергії звукових хвиль від випромінювача в тіло досліджуваного об'єкту здійснюється за допомогою спеціальних гелів, які володіють акустичними властивостями максимально наближеними до акустичних властивостей води.

Ультразвукові хвилі після їх випромінення поширюються у тілі, що досліджується, та відбивається на границях розмежування різних тканин із різними акустичними властивостями.

Уся інформація, отримана від хвиль ультаразвуку, відбитих у зворотньому напрямку, проходить комп'ютерну обробку із подальшою її візуалізацією на моніторах обладнання. Таким чином, завдяки

використанню ультразвукових хвиль створюються тривимірні зображення внутрішніх органів пацієнта.

Сучасні ультразвукові діагностичні апарати здатні працювати у різних режимах діагностики: двовимірному режимі, тривимірному або чотиривимірному. В даному випадку, кількість вимірів вказує на об'ємність отриманого зображення. Найбільш поширеним і, водночас, найпершим варіантом ультразвукової діагностики була двовимірна діагностика. Створені таким способом зображення були представлені у відтінках сірого і представляли собою пласкі зрізи внутрішніх органів людини. При використанні тривимірної та чотирьохвимірної діагностики можливо отримати справжнє тривимірне медичне зображення. При чому в останньому випадку тривимірне зображення створюється в реальному масштабі часу.

1.2.3. Магнітно-резонансна томографія

Магнітно-резонансна томографія – це один із томографічних методів вивчення внутрішньої будови живого організму. В основі даного методу лежить фізичне явище ядерного магнітного резонансу.

Даний метод дозволяє одержати чітке та якісне контрастне зображення різних тканин людського організму. Саме через це його використання широко розповсюджене у медичній галузі, зокрема у візуалізації м′язових тканин, тканин мозку, серця [13].

Порівняно з іншими методами візуалізації, такими як, наприклад, комп'ютерна томографія, магнітно-резонансна томографія є більш поширеною.

Магнітно-резонансна томографія належить до числа тих методів медичного обстеження, які дуже часто знаходять своє застосування у контролі за адекватністю лікування хворого та у різних видах медичної діагностики. На відміну від комп′ютерної томографії та використання рентгенівського опромінення, під час використання даного методу організм людини не зазнає шкідливого впливу іонізуючого

випромінювання. Замість цього формування зображення відбувається під впливом потужного магнітного поля та електромагнітних хвиль із застосуванням подальшої комп′ютерної обробки для одержання більш якісної деталізації, кісток, м′яких тканин та внутрішніх органів організму людини. Під час проведення магнітно-резонансної томографії також часто використовують контрастні речовини для підвищення чіткості кінцевого зображення. З використанням магнітно-резонансної томографії можна виявити та діагностувати патологічні відхилення, які, при використанні інших підходів до медичної візуалізації, спостерігати неможливо.

Методики магнітно-резонансної томографії постійно розвиваються та вдосконалюються. Наразі їх використання дозволяє досягнути високого рівня деталізації, наприклад, одержувати зображення ділянок мозку менш ніж 1 мм завтовшки.

1.3. Формати збереження даних медичних зображень

Дані тривимірного зображення зберігаються у файлах спеціального формату. Зазвичай, формати збереження даних воксельних моделей визначаються виробниками апаратного забезпечення, яке, власне, і дозволяє отримувати ці моделі.

Для того, щоб можливо було якимось чином оперувати даними створеної моделі та здійснювати їхню обробку, не достатньо зберегти лише значення відповідних вокселів. Отримана інформація про вокселі зберігається у файлі лінійно, що зроблено для оптимізації, оскільки у такому випадку не потрібно зберігати координати кожного вокселя. Натомість, необхідно певним способом зберігати інформацію про розмірність моделі, кількість елементів у ній, розмір кожного запису для того, щоб можна було отримати доступ до потрібного вокселя у файлі. Також необхідно знати порядок обходу вокселів та інформацію про орієнтацію моделі у просторі, оскільки різні прилади для сканування можуть записувати дані про модель різним чином, і, відповідно, порядок

вокселів може відрізнятися для тривимірних зображень, отриманих різним апаратним забезпеченням.

Тому, для збереження службової інформації, необхідної для обробки даних моделі програмним шляхом, формати збереження медичних зображень передбачають наявність двох файлів: так званого, header-файлу, для збереження вище згаданої службової інформації, та, власне, файлу у якому зберігаються значення вокселів.

У табл. 1.1 наведено перелік основних полів, що зберігаються у header-файлі формату NIFTI, їхні розмірності в байтах та положення у файлі.

Таблиця 1.1

Структура header-файл формату NIFTI

Тип	Назва	Позиція в файлі (позиція/обсяг)		Призначення
int	sizeof_hdr	0B	4B	Розмір файлу в байтах
char	dim_info	39B	1B	Кодування розмірностей
short	dim[8]	40B	16B	Масив розмірностей
short	datatype	70B	2B	Тип даних, що зберігаються
short	bitpix	72B	2B	Кількість бітів на воксель
float	pixdim[8]	76B	32B	Лінійні розміри для відповідних вимірів
char	xyzt_units	123B	1B	Одиниці виміру для координат
float	slice_duration	132B	4B	Інтервал відображення зрізу
char	descrip[80]	148B	80B	Опис моделі
char	intent_name[16]	328B	16B	Назва або призначення збережених даних

Як було зазначено, вище порядок запису даних вокселів може бути різним, в залежності від того, яке апаратне забезпечення було використане. Це зумовлено тим, яким чином та згідно якого принципу відбувалося сканування досліджуваного об'єкту. Відповідно до цього, для того, щоб можливо було здійснити візуалізацію моделі та виконувати її обробку, необхідно мати змогу однозначно зберігати інформацію про орієнтацію моделі.

Для цього у header-файлах зберігаються додаткові поля, що містять дані, необхідні для афінних перетворень або іншу інформацію, якщо доступ до збережених вокселів передбачається здійснювати іншим методом.

У найпростішому випадку, коли координати вокселів у певному дискретному просторі співпадають із індексами їх лінійного подання у файлі, може бути використане наступне перетворення:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} i \\ j \\ k \end{bmatrix} \circ \begin{bmatrix} \text{pixdim}[1] \\ \text{pixdim}[2] \\ \text{pixdim}[3] \end{bmatrix}, \tag{1.1}$$

де x, y, z – координати вокселя у просторі, i, j, k – індекси положення значення вокселя у файлі, а pixdim – відповідні лінійні розміри вокселя по кожному виміру.

На практиці, для збереження даних тривимірних медичних зображень, зазвичай, використовують стандарти DICOM та NIFTI.

1.3.1. DICOM

DICOM (англ. Digital Imaging and Communications in Medicine) – галузевий стандарт створення, збереження, передачі та візуалізації тривимірних медичних зображень та документів обстежених пацієнтів. DICOM спирається на ISO-стандарт OSI, підтримується основними виробниками медичного обладнання та медичного програмного забезпечення [14].

Стандарт DICOM, що розробляється Національною асоціацією виробників електронного устаткування (National Electrical Manufacturers Association), дозволяє створювати, зберігати, передавати і друкувати окремі кадри зображення, серії кадрів, інформацію про пацієнта, дослідження, обладнання, установи, медичний персонал, що проводить обстеження і т. п. [15]

Стандартом DICOM визначено два інформаційних рівня:

- файловий рівень – DICOM File (DICOM-файл) – об’єктний файл з теговою організацією для подання кадру зображення (або серії кадрів) і супроводжуючої/керуючої інформації (у вигляді DICOM тегів);
- мережевий (комунікаційний) – DICOM Network Protocols (мережевий DICOM-протокол) – для передачі DICOM файлів і керуючих DICOM команд по мережах з підтримкою TCP/IP.

Інформаційна модель стандарту DICOM для DICOM файлу є чотирьохступеневою:

- дані про пацієнта;
- дані про дослідження;
- серія;
- зображення (кадр або серія кадрів).

Файловий рівень стандарту DICOM описує:

- атрибути і демографічні дані пацієнта;
- модель і фірму виробника апарату, на якому проводилося обстеження;
- атрибути медичного закладу, де було проведено обстеження;
- атрибути персоналу, який проводив обстеження пацієнта;
- вид обстеження і дата/час його проведення;
- умови та параметри проведення дослідження пацієнта;

– параметри зображення або серії зображень, записаних в DICOM-файлі;
– зображення, серію або набір серій, отриманих при обстеженні пацієнта;
– іншу інформацію.

1.3.2. NIFTI

Формат NIFTI (Neuroimaging Informatics Technology Initiative) було передбачено в якості заміни раніше широко поширеного формату ANALYZE 7.5. Основна проблема із попереднім форматом була пов'язана з відсутністю належної інформації про орієнтацію моделей в просторі і збережені дані не могли бути однозначно інтерпретовані. Незважаючи на те, що файл був використаний багатьма різними програмами обробки зображень, відсутність інформації по орієнтацію моделі змушували для кожного файлу формату ANALYZE додавати супровідний файл, що описує орієнтацію, наприклад, файл з розширенням .mat. Представники деяких з найбільш популярних програм нейровізуалізації домовилися про формат, який буде включати в себе нову усю необхідну інформацію для візуалізації, що і призвело до появи формату NIFTI. [16]

Згідно цього формату, інформація про тривимірне медичне зображення подається у двох частинах: заголовку, що містить різну метаінформацію про модель (розмір вокселя, орієнтацію моделі у просторі, вид сканування, розмірність моделі та ін.), та основної частини, у якій зберігається інформація про вокселі. Тривимірне зображення може бути подане у вигляді двох файлів, де інформація заголовку винесена у спеціальний файл із розширенням .hdr, а основна частина подана у файлі формату .img. Також є можливість зберігати усю інформацію в одному файлі формату .nii.

Модель має декілька вимірів. Перші три виміри зарезервовані для визначення трьох просторових вимірів – x, y і z – в той час як четвертий вимір зарезервовано для визначення моментів часу – t. Решта вимірів, від

п'ятого до сьомого, призначені для використання в інших цілях. П’ятий вимір може бути використаний для зберігання додаткових параметрів вокселів [17].

1.4. Візуалізація воксельних даних

Для візуалізації тривимірних медичних зображень була розроблена низка методів та алгоритмів. За своїм принципом, всі вони поділяються на три види:

- принцип розкладення на декілька площин (multiplanar reformation);
- рендеринг поверхонь (surface rendering);
- об’ємний рендеринг (volume rendering).

Як буде показано надалі, методи візуалізації можуть бути класифіковані як прямі методи та непрямі методи візуалізації (рендерингу). Методи прямого рендерингу включають в себе прямий об’ємний рендеринг та прямий рендеринг поверхонь, в той час, як непрямий рендеринг включає в себе методи непрямого рендерингу поверхонь [18]. Слід зазначити, що перед безпосереднім збереженням та візуалізацією даних тривимірне медичне зображення має пройти попередню обробку [19].

Взагалі, стратегія непрямого рендерингу тривимірних зображень полягає у створенні полігональної моделі поверхні для заданої воксельної моделі даних. Полігональні моделі поверхонь можуть бути ефективно візуалізовані і такі моделі легко керовані програмним шляхом. Отримані полігональні моделі поверхонь непрозорі і є порожніми всередині, і, таким чином, лише імітують суцільні воксельні моделі, для яких вони були згенеровані. Відповідно, поверхня, отримана внаслідок такої генерації не я точною, а лише надає апроксимацію реальної поверхні початкової воксельної моделі.

Таким чином, можна зробити висновок, що уся складність підходу непрямого рендерингу полягає у пошуку полігональної моделі, яка

максимально точно відтворює реальну поверхню. Після знаходження такої поверхні, її відображення є тривіальною задачею.

На відміну від непрямого рендерингу, пряма візуалізація – це безпосереднє відображення вокселів на пікселі двовимірної площини зображення. Відмінність від попереднього підходу полягає в тому, що прямий рендеринг надає ширші можливості для візуалізації, проте є набагато складнішим та витратнішим.

Воксельні дані можна розглядати як напівпрозорі, і користувач вирішує, які частини об’єкта повинні бути непрозорими або прозорими. Остаточне 2D-зображення обчислюється шляхом проектування, у порядку видимості, вокселів на площину зображення та поступового додавання кольору та прозорості вокселів до кінцевого пікселя [20].

Окрему увагу слід приділити візуалізації фрагментів тривимірних зображень, зокрема, візуалізації зрізів воксельних моделей. Технологія отримання зрізів є поширеним методом візуалізації. Вони відображають дані про об’єм, фактично, вокселі відображаються на пікселі двовимірної площини.

Варто відзначити, що головною метою при візуалізації воксельних моделей даний є інтеграція різних методів та їх спільне використання, щоб в результаті представити ці дані якнайкраще.

Проте необхідно пам’ятати, що найбільш правильний метод з точки зору фізичного реалізму не є найбільш оптимальним з точки зору розуміння та представлення даних. Крім того, для відтворення та відображення тривимірних значень завжди потрібно створювати двовимірні зображення, що передбачає проекцію та втрату даних, оскільки в такому разі відкидається один вимір.

1.4.1. Розкладення просторових даних в декількох площинах

Розкладення просторових даних в декількох площинах – це техніка візуалізації тривимірних медичних зображень, що полягає в отриманні та відображенні набору двовимірних зрізів воксельних даних із моделі.

Зазвичай використовуються ортогональні зрізи тривимірного зображення, хоча можуть бути використані і зрізи, отримані під довільним кутом.

Хоча даний метод є, по суті, двовимірною візуалізацією, він має ряд переваг, такі як простота, достатній рівень наочності, висока швидкодія та відсутність втрати інформації внаслідок візуалізації.

Даний спосіб представлення моделі через її зрізи, частіше за все, розглядається як різновид непрямого рендерингу воксельної моделі. Зрізи тривимірного зображення можуть бути як ортогональними, так і довільно орієнтованими. Отримання ортогональних зрізів, тобто зрізів, перпендикулярних до координатних осей *x, y, z* є тривіальною задачею, в той час як отримання довільно орієнтованих зрізів, в загальному випадку, потребує спеціальних методів. Найпоширенішим та найдоступнішим способом візуалізації таких зрізів є використання можливостей рендерингу у сучасних графічних процесорах, зокрема, тривимірних текстур. При використанні такої техніки, значення кожного пікселю двовимірного зображення зрізу обчислюється в результаті застосування інтерполяції значень сусідніх вокселів (тобто, кожного елемента тривимірної текстури).

Приклад візуалізації в декількох площинах наведено на рис. 1.1.

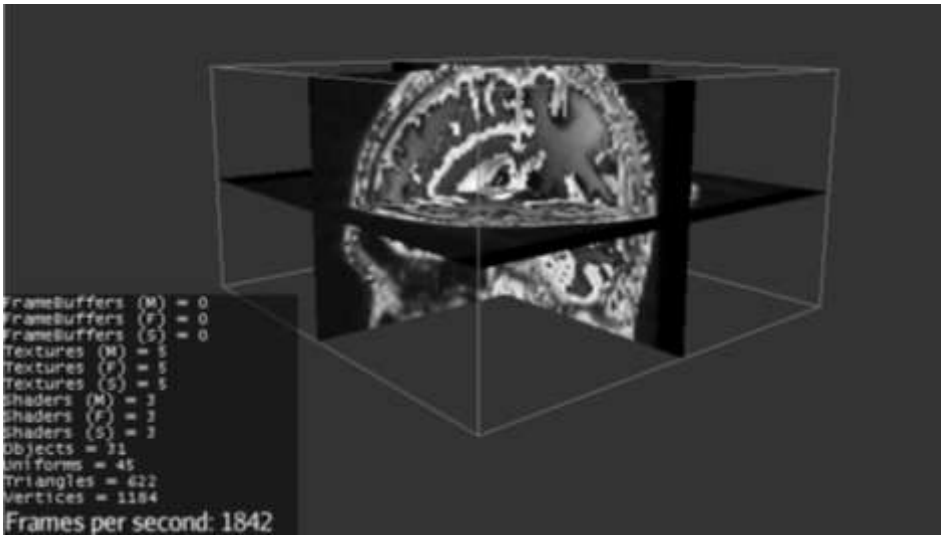


Рис. 1.1. Воксельна модель голови людини, представлена трьома ортогональними зрізами

1.4.2. Рендеринг поверхонь

Рендеринг поверхонь – ще один клас методів для відображення тривимірних зображень. Умовно, рендеринг поверхонь поділяється на два типи: непрямий рендеринг та прямий рендеринг поверхонь.

Непрямий рендеринг поверхонь полягає в моделюванні структури поверхні воксельної моделі шляхом сегментації. Для прямого рендеренигу візуалізація виконується безпосередньо на основі воксельних даних, проміжні кроки для отримання геометричного представлення моделі (наприклад, у вигляді полігональної сітки) відсутні.

Для генерації поверхонь широко використовується алгоритм marching cubes, який, фактично, є стандартним для перетворення набору вокселей в полігональну модель.

Marching cubes (з англ. – «крокуючі куби») – алгоритм в комп'ютерній графіці для обробки полігональної сітки ізоповерхні тривимірного скалярного поля (частіше званої сіткою вокселів). Вхідними даними для даного алгоритму є набір вокселів тривимірного зображення, яке необхідно візуалізувати. Результатом є готова полігональна модель, що представляє вихідне тривимірне зображення. Даний алгоритм широко використовується в медицині, наприклад, у комп’ютерній і магнітно-резонансної томографії [21].

1.4.3. Об’ємний рендеринг

Об’ємний рендеринг – техніка, яка використовується для отримання плаского зображення (проекції) тривимірного дискретного набору даних (даних воксельної моделі).

Вхідним набором даних є тривимірне зображення, отримане при комп’ютерній томографії або магнітно-резонансній томографії. Зазвичай шари, із яких складається тривимірне зображення, мають рівну товщину (наприклад, 1 мм) і рівну кількість вокселів на кожен шар. Таким чином, вхідними даними є регулярна сітка вокселів, де кожен воксел відповідає

усередненим значення (температура, щільність матеріалу) в даній точці тривимірного об’єкту.

При застосуванні прямого об’ємного рендерингу, тривимірне зображення візуалізується цілком, без жодної генерації геометричного представлення поверхні моделі.

Прямий об’ємний рендерер зіставляє значенню кожного вокселя колір і прозорість. Це робиться за допомогою передавальної функції, яка може здаватися кусочно-лінійною функцією або таблицею значень. Після цього отримане RGBA значення виводиться в кадровий буфер. В результаті промальовування всього обсягу виходить цілісне зображення.

На практиці використовується ціла низка методів та способів прямого рендерингу воксельних моделей [18, 22, 23]:

- програмний рейкастинг та рейкастинг із використанням графічних процесорів;
- метод кидання сніжків;
- метод shear warp;
- shell-рендеринг;
- використання тривимірних текстур;
- інші методи.

Найбільш поширеним є спосіб візуалізації воксельних моделей із використанням тривимірних текстур. Вся модель завантажується у пам’ять графічного процесора у вигляді такої текстури. Потім створюється набір полігонів, перпендикулярних до напрямку перегляду моделі, і на ці полігони накладається дана тривимірна текстура моделі відповідно до заданих текстурних координат. Композиція усіх текстурованих полігонів відображається шляхом побудови набору проекцій на площину перегляду, і таким чином, відбувається візуалізація воксельної моделі.

1.5. Методи растеризації воксельних примітивів

В деяких випадках при вирішенні спеціалізованих задач буває недостатнім отримати воксельну модель і візуалізувати її або побудувати її полігональну модель. У медичній галузі в ході дослідження та вивчення отриманих тривимірних зображень може знадобитися візуалізація не лише усього об’єкту дослідження, але і його складових та окремих частин, а також постає проблема вивчення зрізів моделі.

Щоб отримати зріз тривимірного зображення, необхідно визначити належність вокселів моделі до площини зрізу, а потім провести їх візуалізацію. Як було зазначено раніше, візуалізація зрізів, отриманих під довільними кутами перетину воксельної моделі є нетривіальною задачею, оскільки не можна однозначно визначити належність конкретного вокселя площині зрізу, а на моделях великого об’єму необхідно застосовувати спеціальні алгоритми розтину воксельної моделі для уникнення повного перебору усіх вокселів.

Розвиток тривимірних скануючих пристроїв, таких як пристрої магнітно-резонансної томографії, впровадження програмного забезпечення і систем візуалізації об’ємних моделей, а також розвиток спеціалізованих пристроїв 3D-візуалізації призвели до появи алгоритмів, які виконують растеризацію у тривимірному дискретному просторі. Сімейством таких алгоритмів є, так звані, ткацькі алгоритми.

Суть ткацьких алгоритмів растеризації полягає в наступному. На першому етапі виконується растеризація певної кривої, яку називають основною (base-curve). На другому етапі растеризується інша крива, яка називається головною (master-curve). Візуалізація поверхні полягає у послідовній візуалізації головної кривої, яка поступово зміщується уздовж основної. Таким чином буде візуалізована певна частина поверхні воксельної моделі.

Замість растеризації кривих у загальному випадку, можна растеризувати лінії, і таким чином візуалізована частина поверхні буде

частиною площини. Використовуючи ткацькі алгоритми можна будувати і візуалізовувати поверхні зрізів медичних зображень.

Загалом, задача растеризації зрізів тривимірних медичних зображень, зводиться до задачі растеризації полігонів. Тому також існує ряд методів, що базуються на методах та алгоритмах для растеризації примітивів у двовимірному просторі.

1.6. Висновки

В рамках даного дослідження, були вивчені основні характеристики тривимірних медичних зображень, шляхи їх одержання, їх структура та формати даних для їх збереження. Були досліджені різні методи та алгоритми візуалізації та растеризації воксельних моделей, їх переваги та недоліки.

Основною проблемою при обробці тривимірних зображень є знаходження компромісу між точністю отриманих результатів та часом виконання обробки. Зокрема, було встановлено, що для роботи із воксельними моделями часто використовуються спеціалізовані апаратні засоби.

Також було виявлено, що растеризація та візуалізація довільно орієнтованих зрізів тривимірних медичних зображень залишається актуальною задачею.

Виходячи із результатів проведеного дослідження, було запропоновано розробити власний метод, який базується на результатах раніше проведених досліджень, проте дозволяє оптимізувати виконання растеризації та, водночас, забезпечити достатню точність для представлення зрізів воксельних моделей даних під довільними кутами.

2. РОЗРОБЛЕННЯ МЕТОДУ РАСТЕРИЗАЦІЇ ЗРІЗІВ ТРИВИМІРНИХ МЕДИЧНИХ ЗОБРАЖЕНЬ

2.1. Можливості оптимізації методів растеризації

Методи растеризації зрізів тривимірних медичних зображень можуть оцінюватися за двома основними показниками: швидкодією та точністю отриманих результатів.

Кожен із цих показників є, безсумнівно, важливим, але на практиці їх поєднання викликає суттєві технічні труднощі. Зазвичай, для того, аби провести якісну растеризацію зрізу моделі, необхідно задіяти значні обчислювальні ресурси. З іншого боку, обробка моделей повинна виконуватись за прийнятний час. Тому проблема растеризації зрізів медичних зображень потребує пошуку компромісних рішень, які здатні задовольнити поставлені вимоги.

Як було зазначено раніше, методи растеризації зрізів базуються на методах растеризації полігонів у двовимірному просторі, проте розширюють їх для можливості використання у тривимірному. Варто відмітити, що растеризація у тривимірному просторі являє собою значно складніший процес, ніж у двовимірному. Одні методи та алгоритми були розроблені з метою забезпечення максимальної точності растеризації, інші – для отримання максимальної швидкодії. Через це, усі можливі оптимізації спрямовані на досягнення максимального ефекту для цих двох показників.

За часів, коли використання потужних комп’ютерів було обмежене, як і можливості тодішніх графічних процесорів, розроблювалися та пропонувалися спеціалізовані апаратні засоби, здатні виконувати обробку та візуалізацію великих тривимірних зображень та їх фрагментів, зокрема, зрізів [24, 25]. Такі системи мали складну архітектуру, були вузько спеціалізованими, а їх практичне використання було значно ускладненим. Проте із розвитком комп’ютерних відеокарт, можливості рендерингу воксельних моделей стрімко зросли. Стали доступними нові методи та

способи візуалізації, які можна реалізувати засобами графічних процесорів (наприклад, використання тривимірних текстур).

В рамках даного дослідження були розглянуті існуючі методи та алгоритми растеризації площин та полігонів у тривимірному просторі та можливості їхньої оптимізації.

Кауфман [26, 27] пропонував свій scan-conversion алгоритм для растеризації полігонів. Запропонований алгоритм був заснований на принципі scanline-растеризації [28] у двовимірному просторі і розширений для використання у тривимірному просторі. Його можна використовувати для побудови зрізів воксельної моделі.

Принцип полягає у растеризації контура полігону (контура зрізу) і поступове його заповнення, растеризуючи паралельні лінії, що з’єднують протилежні точки отриманого контура. На рис. 2.1 зображено процес такої растеризації.

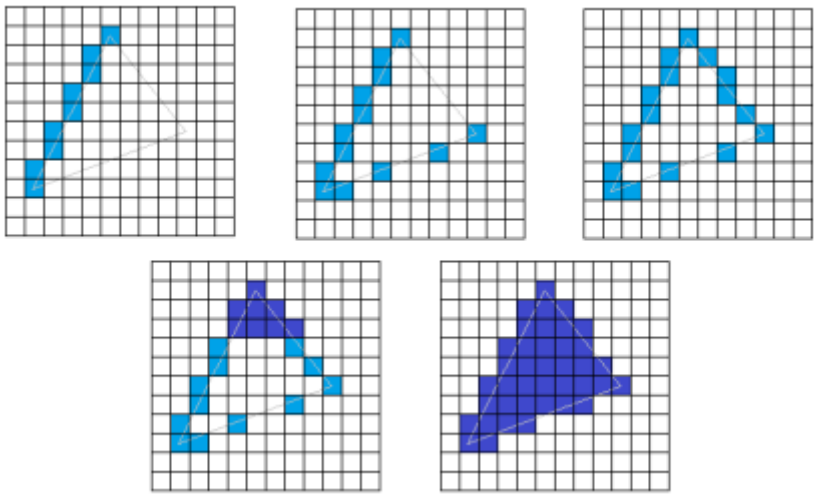


Рис. 2.1. Scanline-растеризація полігону

Запропонований метод здатен забезпечити високу швидкодію растеризації та достатню точність. Даний метод можна реалізувати таким чином, щоб в алгоритмі растеризації використовувались лише цілочисельні операції.

Проте недоліками розглянутого методу є те, що координати кожного вокселя необхідно обчислювати окремо, і жодним чином не

використовується той факт, що растеризація, по суті, паралельних ліній може бути виконана один раз, а інші лінії можуть бути растеризовані паралельним перенесенням [29]. До того ж, метод потребує додатково ініціалізації, під час якої має бути здійснена растеризація контурів полігону та визначені пари точок для подальшого їх з'єднання, що в тривимірному просторі є не зовсім тривіальною задачею [30].

На останок варто зазначити, що запропонований метод може працювати для довільних полігонів. Така універсальність може виявитися корисною при вирішення специфічних задач растеризації, проте для растеризації пласких зрізів тривимірних медичних зображень представлена можливість буде збитковою та може ускладнити процес растеризації зайвими операціями із моделлю.

Інший підхід, що може бути застосований при растеризації зрізів воксельної моделі, ґрунтується на використанні ткацьких технологій [31, 32]. Основна ідея полягає в тому, що одна растеризована лінія (master-лінія) копіюється уздовж іншої растеризованої лінії – base-лінії – із певним зсувом. Таким чином, відбувається растеризація площини, яка містить master-лінію та base-лінію.

Одним із алгоритмів, що використовують таку техніку є ткацький алгоритм точної растеризації [33], використання якого дозволяє отримати достатньо точне представлення зрізу воксельної моделі. При цьому кількість обчислень оптимізується за рахунок обчислення спеціальних ланцюжків для відрізків ліній, що утворюють площину зрізу [34].

Якщо представити задану лінію наступним чином:

$$L = \left\{ (x, y) \in R^2 : y = \frac{p}{q} \cdot x + e \right\}, \tag{2.1}$$

де x, y – координати довільної точки лінії у просторі, а p, q, e – задані коефіцієнти, то дана лінія в дискретному просторі може бути подана як:

$$Dig(L) = \left\{ (i, y) \in Z^2 : y = \left\lfloor \frac{p}{q} \cdot i + e \right\rfloor \right\}, \tag{2.2}$$

де i, y – координати довільної точки лінії у дискретному просторі.

Згідно запропонованого алгоритму, необхідно обчислити наступну послідовність – ланцюжок зсувів:

$$c_i = y(i) - y(i-1). \tag{2.3}$$

Доведено, що такий ланцюжок повторюються із певною періодичністю [35], а отже для проведення растеризації площини достатньо його обчислення, а потім послідовного копіювання, що дозволяє зменшити кількість необхідних обчислень для растеризації усієї площини. Таким чином, растеризація ліній спрощується, і необхідно лише знайти початкову точку $(\lfloor y(i_0) \rfloor, i_0)$ та відповідний ланцюжок зсувів c. Значення функції обчислюється за наступною формулою:

$$\lfloor y(i_0 + k) \rfloor = \lfloor y(i_0) \rfloor + \sum_{i=1}^{k} c_{i_0 + i}, \tag{2.4}$$

де k – номер поточного елемента лінії.

Також існують методи, що дозволяють робити зрізи, вокселі яких повністю покривають площину зрізу [36]. Такі зрізи складаються із усіх вокселів, що перетинаються заданою площиною, що є необхідною вимогою у деяких випадках, але товщина такого зрізу може бути більша за один воксель.

В результаті дослідження та аналізу попередніх результатів пропонується власний метод, що подібний до ткацьких алгоритмів растеризації та дозволяє отримувати зрізи воксельної моделі одиничної товщини під довільним кутом. В основі методу лежить принцип, який використовується в ткацьких алгоритмах растеризації, проте даний метод використовує лише цілочисельну арифметику. Отримані зрізи воксельних моделей не можуть вважатися точними, проте надають достатню

апроксимацію, для того, щоб використання методу було доцільне в більшості практичних задач.

2.2. Основні поняття

Воксель – частина дискретного простору, яка представляє собою певне значення і є складовою воксельної моделі. В контексті даної роботи розглядається як одиничний куб.

Будемо вважати, що воксельна модель – це сукупність вокселів, які дотикаються своїми гранями одне до одного та формують структуру у вигляді паралелепіпеда. У даній структурі немає порожнин, тобто кожна грань внутрішнього вокселя дотикається до граней сусідніх вокселів. Модель має цілочисельні довжину, висоту та ширину.

Кожен воксель у такій моделі має власні, також цілочисельні координати:

$$x = \overline{0..L-1}, \tag{2.5}$$

$$y = \overline{0..H-1}, \tag{2.6}$$

$$z = \overline{0..W-1}, \tag{2.7}$$

де L – довжина моделі, H – висота моделі і W – ширина моделі, відповідно.

Початок прямокутної системи координат, в якій розглядається модель, співпадає з позицією вокселя з координатами $(0,0,0)$. Грані моделі належать площинам *Oxy*, *Oxz*, *Oyz*.

Під растеризацією площини зрізу розуміється знаходження множини усіх вокселів, або точніше, їхніх координат, що утворюють задану площину.

Для опису зв’язків між вокселями застосовується поняття зв’язності.

Зв’язність при растеризації воксельних моделей – це деяка характеристика, що описує, яким чином вокселі тривимірного зображення пов’язані між собою.

Розрізняють три, принципово різні, типи зв’язку:

- два сусідніх вокселі мають спільну вершину;

– два сусідніх вокселі мають спільне ребро;

– вокселі мають спільну грань.

Відповідно до такого розподілу, визначають наступні види зв’язності вокселів:

– 6-ти зв’язність;

– 18-ти зв’язність;

– 26-ти зв’язність.

У випадку 6-ти зв’язності, два вокселі вважаються пов’язаними між собою, якщо вони мають спільну грань.

Якщо вокселі моделі є 18-ти зв’язними, то це означає, що два сусідніх вокселі повинні мати спільну грань або ребро. В цьому випадку, якщо дотикаються вершинами, то вони не вважаються пов’язаними між собою.

26-ти зв’язні вокселі можуть дотикатися гранями, ребрами та вершинами.

Отримані унаслідок растеризації вокселі можуть представляти площину зрізу по різному. Для опису такого представлення використовується поняття покриття та прозорості.

Покриттям (з англ. – cover) називають множину зв’язних вокселів певної неперервної площини, де кожна точка такої площини належить одному із вокселів покриття. Покриття називають мінімальним, якщо жодна із підмножин вокселів цього покриття не є, в свою чергу, покриттям площини. Суперпокриттям (supercover) площини називають множину усіх вокселів, які перетинаються цією площиною.

Покриття вважається прозорим для певної множини зв’язних вокселів, якщо така множина вокселів проходить наскрізь через задане покриття. Наприклад, для прямої, яка растеризована множиною 6-ти зв’язних вокселів, будь-яке покриття є непрозорим [37].

2.3. Ткацькі алгоритми растеризації

Ткацькі алгоритми для растеризації воксельних моделей – це сімейство алгоритмів, які наслідують принцип ткацьких технологій [38].

Припустимо, що необхідно виконати растеризацію певної ділянки площини. Згідно даної технології, для виконання такої растеризації необхідно обрати дві непаралельні лінії, що лежать у цій площині: master-лінію та base-лінію.

Растеризація заданої площини відбувається шляхом копіювання растеризованої master-лінії уздовж base-лінії. Приклад такої растеризації наведено на рис. 2.2.

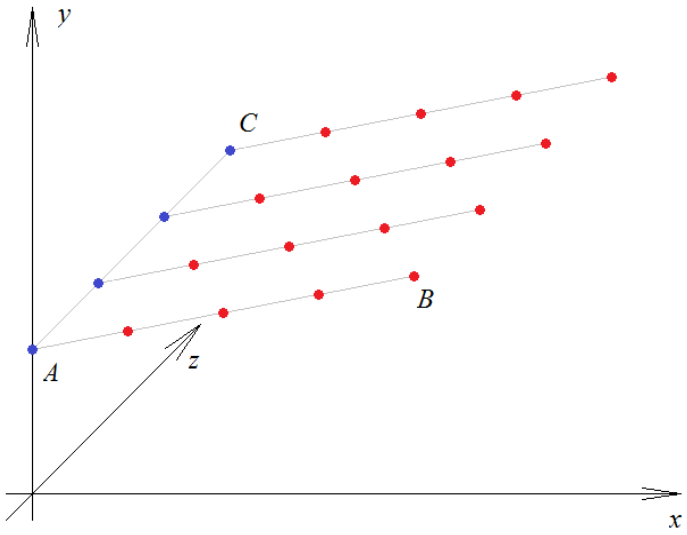


Рис. 2.2. Растеризація площини ткацьким алгоритмом

Нехай master-лінія задана відрізком *AB*, а base-лінія – відрізком *AC*, і, таким чином, точка *A* – спільна точка для обох відрізків. Оскільки вокселі повинні мати цілочисельні координати, то дані прямі розглядаються у дискретному просторі, і, відповідно, координати усіх точок, отриманих при растеризації, будуть цілочисельні.

Спочатку отримана множина точок, що належать відрізку *AB*. Нехай, це буде деяка множина *M*. Далі, рухаючись уздовж точок, що належать відрізку *AC*, відбувається копіювання точок множини *M* із певним зсувом, і, таким чином, відбувається растеризація площини *ABC*.

Різні види ткацьких алгоритмів відрізняються між собою тим, яким чином виконується растеризація master-лінії та base-лінії.

В загальному випадку, можливо використовувати алгоритми растеризації довільних кривих, замість алгоритмів растеризації ліній. В такому випадку, результатом буде растеризація не пласкої площини, а деякої поверхні. В контексті даної роботи, в результаті необхідно отримати плаский зріз воксельної моделі, тому алгоритми растеризації кривих не будуть розглядатися.

2.4. Метод растеризації на основі ткацьких технологій

2.4.1. Основні вимоги

Для площини зрізу, що буде отримана запропонованим у даній роботі методом, поставлені наступні вимоги:

- вокселі даної площини мають задовольняти вимогам 26-зв'язності (тобто можуть дотикатися одне до одного гранями, ребрами або вершинами) [37];
- площина має бути суцільною, тобто не має містити дірок;
- растеризована площина не має містити спотворень.

Даний метод має використовувати цілочисельну арифметику в обчисленнях, що має забезпечити приріст швидкодії. В такому випадку, растеризація площини зрізу не може вважатися абсолютно точною, проте результат має забезпечити непогану апроксимацію, що є задовільним у більшості випадків.

2.4.2. Принцип роботи

Суть способу полягає у тому, щоб растеризувати площину зрізу за допомогою паралельного перенесення фрагментів master-лінії уздовж base-лінії, як в ткацьких алгоритмах растеризації. Для растеризації обох типів ліній пропонується використати алгоритм Брезенхема для побудови ліній [39].

Даний алгоритм є швидким алгоритмом растеризації та дозволяє використовувати лише цілочисельну арифметику, що є безумовною

перевагою. Також, алгоритм Брезенхема для побудови ліній у двовимірному просторі можна розширити для використання у тривимірному просторі [38, 40]. Проте, якщо обмежитися растеризацією ліній у площинах, що ортогональні до координатних осей, то можна використовувати оригінальну версію для двовимірного випадку.

Сам по собі процес растеризації площини – це процес знаходження координат усіх вокселів, що належать даній площині або вважаються такими. Для покращення ефективності та збільшення швидкодії необхідно звести кількість обчислень до мінімуму. Використання алгоритму Брезенхема дозволить провести растеризацію зрізу, застосовуючи лише дії із цілими числами. До того ж, така растеризація має вийти достатньо точною.

Одна ітерація алгоритму Брезенхема для master-лінії – це, по суті, знаходження двох координат для одного вокселя. Третя координата знаходиться на поточній ітерації алгоритму Брезенхема для base-лінії, і є однаковою для усіх вокселів master-лінії.

В загальному випадку, зріз воксельної моделі може бути представлений однією з чотирьох фігур (трикутником, чотирикутником, п’ятикутником або шестикутником), відрізком або однією точною. На рис. 2.3 представлені усі можливі випадки.

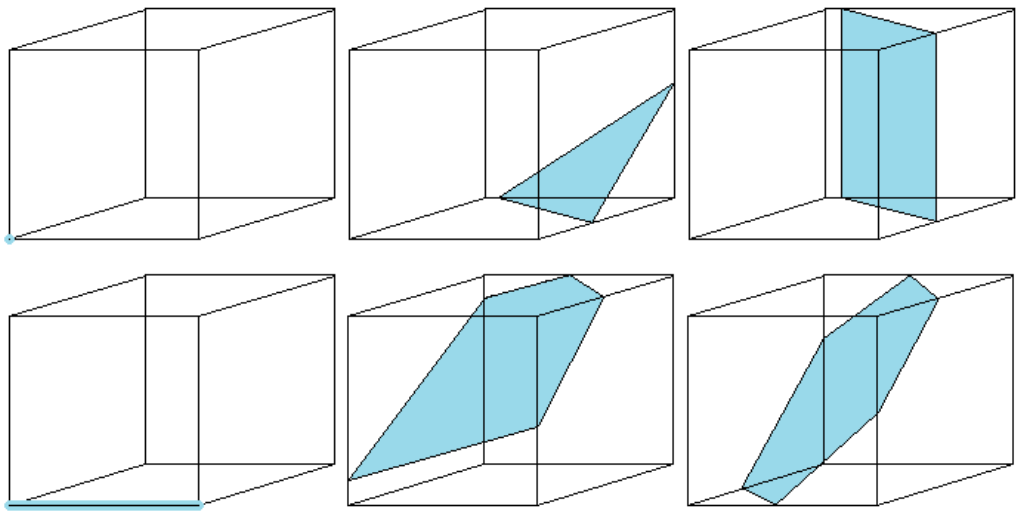


Рис. 2.3. Можливі форми зрізів моделі

Площина зрізу, разстеризацію якого необхідно провести, може задаватися загальним рівнянням площини:

$$ax + by + cz + d = 0, \tag{2.8}$$

де a, b, c та d – задані коефіцієнти.

Для спрощення викладення та без втрати загальності, припустимо, що площина зрізу задається трьома точками, які лежать на осях Ox, Oy, Oz та мають цілі координати. Також для спрощення, будемо вважати, що координати цих точок не можуть бути від'ємними. Позначатимемо ці точки як P_1, P_2, P_3.

Для початку розглянемо підхід, коли растеризація зрізу відбувається алгоритмом Брезенхема без копіювання master-лінії. На рис. 2.4 зображено приклад такої растеризації.

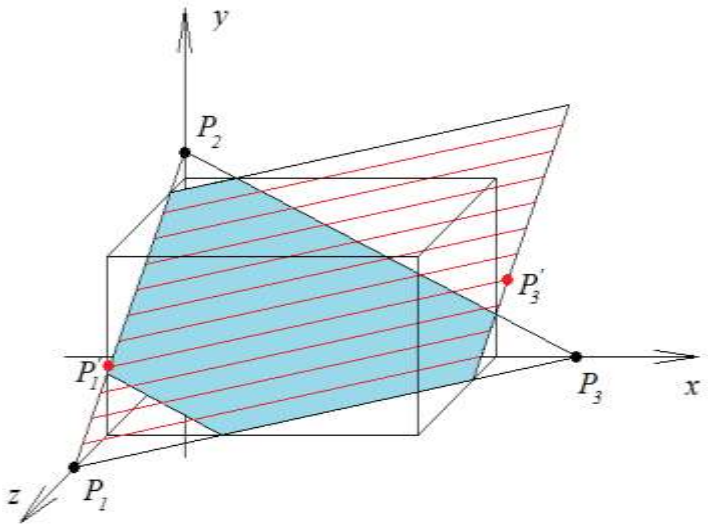


Рис. 2.4. Растеризація зрізу із використанням алгоритму Брезенхема

Нехай, відрізок P_1P_3 обрано як master-лінію, а P_1P_2 – base-лінія. На кожному кроці растеризаці P_1P_2 накопичуються зсуви Δy та Δz по координатах y та z, відповідно, а також здійснюється растеризація поточного відрізка, паралельного до P_1P_3 із врахуванням отриманих зсувів. Таким чином, при проході уздовж P_1P_2, виконується одна растеризація лінії алгоритмом Брезенхема від точки P_1' до точки P_3', де

$$P_1' = P_1 + (0; \Delta y; \Delta z), \tag{2.9}$$

$$P_3' = P_3 + (0; \Delta y; \Delta z). \tag{2.10}$$

Слід зазначити, що при даному підході необхідно перевіряти границі воксельної моделі, щоб не вийти за її межі.

У результаті буде отримана площина, утворена послідовною растеризацією паралельних ліній, що задовольняє початковим вимогам, проте, очевидно, що даний підхід можна значно оптимізувати, якщо правильно обирати master-лінію та base-лінію, проводити растеризацію master-лінії лише один раз та відмовитись від перевірки границь на кожній ітерації, застосувавши певні прийоми.

Пропонується, що при однократній растеризації master-лінії одночасно будуть визначені та збережені ліві та праві границі для наступної растеризації необхідних фрагментів master-лінії. Як буде показано надалі, растеризацію таких фрагментів можна буде програмно здійснювати в простому циклі із лічильником, де в тілі циклу буде застосовуватись єдина операція додавання.

Таким чином, описаний вище метод растеризації буде складатись із двох етапів: етапу ініціалізації, на якому буде здійснюватись растеризація master-лінії та визначення границь, та основного етапу, на якому буде виконано растеризацію фрагментів master-лінії, що визначаються знайденими границями.

2.4.3. Етап ініціалізації

Припустимо, що необхідно растеризувати зріз воксельної моделі, зображений на рис. 2.5. На даному рисунку зріз представлений у вигляді шестикутника, воксельну модель представлено паралелепіпедом, а площину зрізу задано точками P_1, P_2, P_3.

Перед початком растеризації площини зрізу необхідно виконати ініціалізацію.

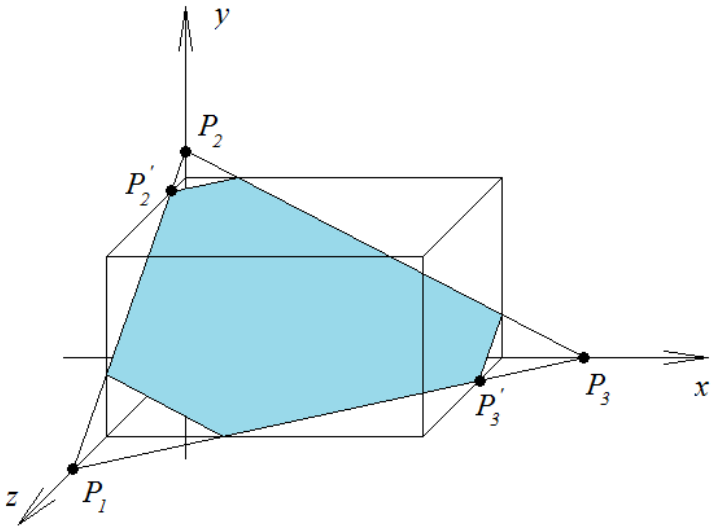


Рис. 2.5. Шестикутний зріз воксельної моделі

На етапі ініціалізації обирається master-лінія та base-лінія, проводиться растеризація master-лінії та обчислюються границі для фрагментів master-лінії, які будуть копіюватися при растеризації площини.

Master-лінія та base-лінія обираються наступним чином.

Для вибору master-лінії необхідно обрати один із відрізків, початком якого буде одна з точок P_1, P_2, або P_3, що належать координатним осям, а кінцем має бути одна із вершин фігури зрізу. При цьому, відрізок має належати одній із площин *Oxy*, *Oxz*, *Oyz*, кут між обраним відрізком та відповідною віссю має бути більше або дорівнювати 45°. При обраному куті, що менше 45° можлива ситуація, коли растеризовані фрагменти master-лінії будуть перекривати одна одну.

Обраний відрізок, не може бути меншим за найменшу із розмірностей моделі, оскільки паралельне перенесення растеризованої master-лінії має покривати усю площину зрізу.

Серед відрізків, які задовольняють вище описані критерії, необхідно обрати найкоротший, який і буде прийнято за master-лінію. Master-лінія обирається найкоротшою із можливих, оскільки таким чином зменшується кількість більш обчислювально-витратних операцій, як буде показано далі. Отже, нехай відповідним відрізком для master-лінії буде P_1P_3'.

Відрізок base-лінії буде лежати в координатній площині, ортогональній до площини master-лінії, причому, його початок буде співпадати із початком відрізка master-лінії. Base-лінія повинна задовольняти таким умовам, оскільки інакше фрагменти master-лінії, при копіюванні будуть перетинатися, що зазначалося вище. Таким чином, відрізком, що відповідатиме base-лінії, буде P_1P_2'.

Після вибору master-лінії та base-лінії, проведемо растеризацію master-лінії, використовуючи алгоритм Брезенхема. Під час растеризації master-лінії визначаються границі A_L та A_R (ліві та праві границі, відповідно) фрагментів, що будуть копіюватися при растеризації площини, та ланцюжок зсувів C.

Розглянемо рис. 2.6, де відображено результат такої растеризації для точок $P_1(0,0,8)$ та $P_3'(9,0,2)$.

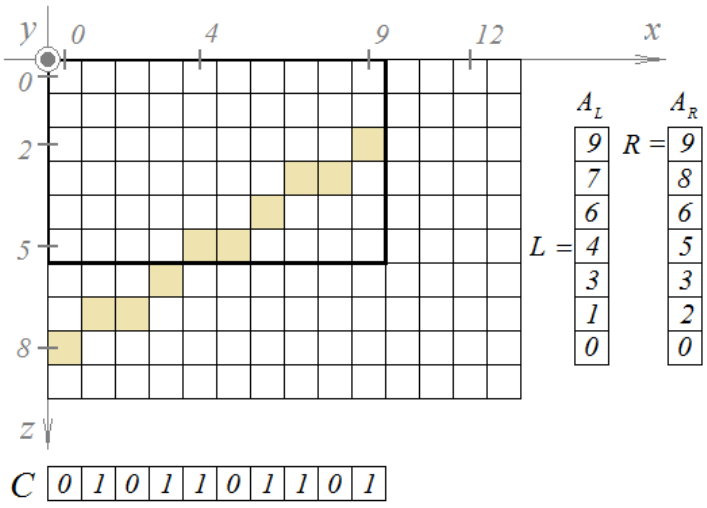


Рис. 2.6. Растеризація master-лінії

На рисунку зображена, растеризована уздовж осі *Ox*, master-лінія та заповнені значення масивів A_L, A_R та C. Масив C – це ланцюжок зсувів, кожен елемент якого дорівнює 0 або 1, відповідно до того, чи мав місце зсув на поточній ітерації, чи ні. Масив A_L, в даному прикладі, містить ті координати x, що відповідають зсувам у ланцюжку C, а масив A_R – ті координати x, що відповідають зсувам, які б були отримані при

растеризації master-лінії у зворотньому напрямі.

Також на етапі ініціалізації встановлюються значення змінних *L* та *R*, які задають початкові границі, в межах яких буде растеризовано перший фрагмент master-лінії. Змінна *L* відповідає елементу A_L, який перший потрапляє в межі моделі, а змінна *R* – останньому елементу A_R.

2.4.4. Основний етап

Після етапу ініціалізації проводиться основний етап, який полягає у копіюванні фрагментів отриманої master-лінії уздовж base-лінії, яка також растеризується алгоритмом Брезенхема.

При копіюванні використовуються наступні формули:

$$x_r = i,\ i = \overline{L..R}, \tag{2.11}$$

$$y_r = y_b, \tag{2.12}$$

$$z_r = z_{r-1} - C_i, \tag{2.13}$$

$$z_0 = \begin{cases} W, & if\ z_b \geq W \\ z_b, & if\ z_b < W \end{cases}, \tag{2.14}$$

де z_b та y_b – координати поточного вокселя при растеризації base-лінії, а x_r, y_r, z_r – координати вокселя із фрагменту master-лінії.

Змінні *L* та *R* визначають межі, в яких буде виконуватись копіювання фрагменту master-лінії, і є індексами масиву *C*. Їх значення змінюється під час растеризації base-лінії при зміні координати *z*. Змінна *L* буде змінювати своє значення на попереднє в масиві A_L, доки не досягне першого елементу (нульового). Змінна *R*, буде залишатися постійною, доки

$$z_b \geq S, \tag{2.15}$$

де *S* – кількість зсувів, а потім буде змінювати своє значення, аналогічно змінній *L*.

На рис. 2.7 зображено хід растеризації площини зрізу.

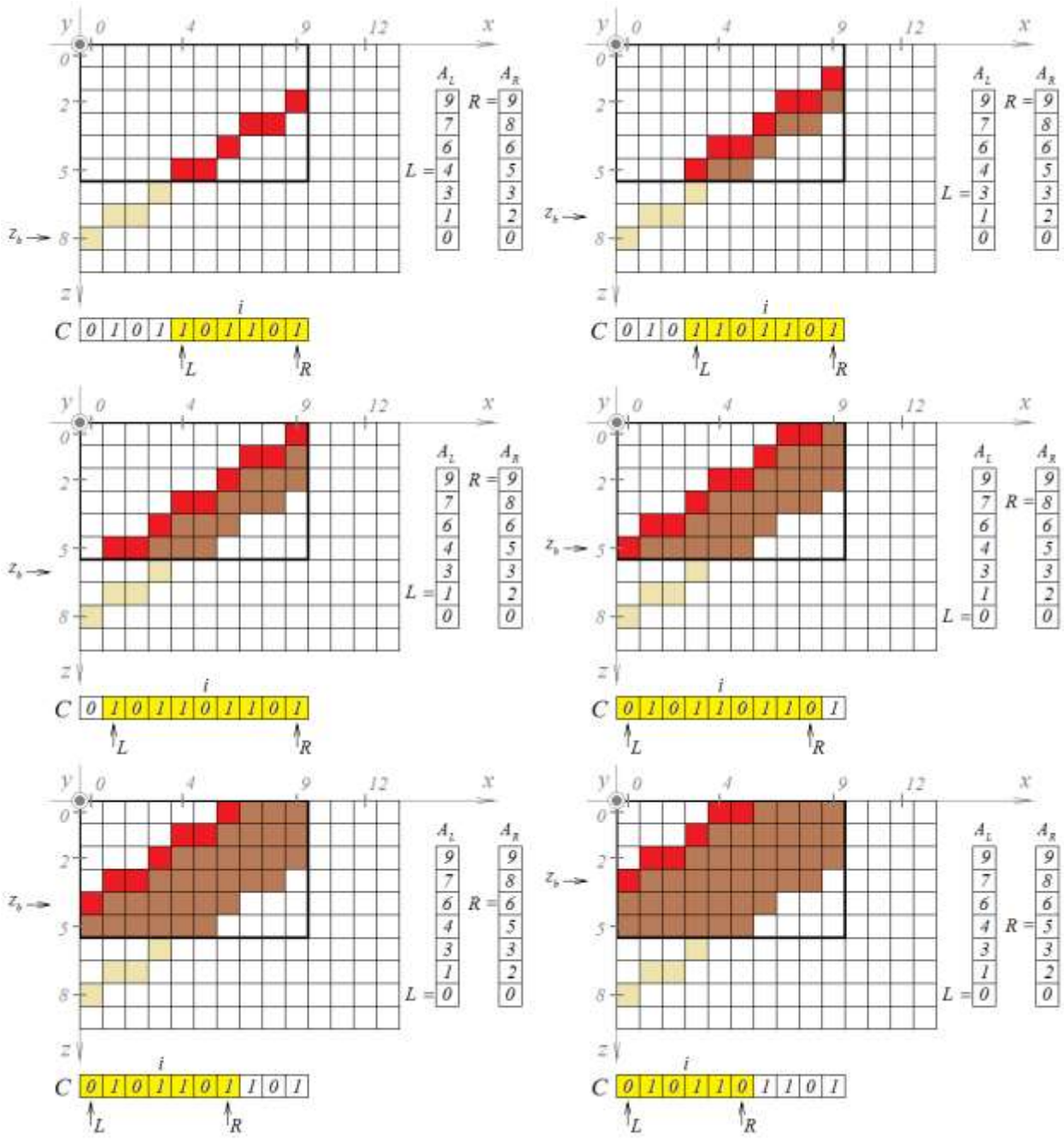


Рис. 2.7. Хід растеризації зрізу (вид згори на площину *Oxz*)

2.5. Висновки

Запропонований метод растеризації дозволяє отримувати зрізи воксельних моделей даних під довільними кутами. Отримані зрізи мають одиничну товщину, та достатньо точно відтворюють площину зрізу. Розроблений метод растеризації використовує лише цілочисельну арифметику та мінімізує кількість обчислень в циклах растеризації фрагментів master-лінії, що робить використання методу більш ефективним для растеризації зрізів воксельних моделей.

3. ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АЛГОРИТМІВ РАСТЕРИЗАЦІЇ ЗРІЗІВ

3.1. Вимоги до програмної реалізації

Для методів растеризації зрізів воксельних моделей даних характерна певна специфіка, пов’язана із особливістю оброблюваних даних. Через це алгоритми, що реалізують відповідні методи також мають певні особливості роботи та реалізації.

В рамках даного дослідження було запропоновано метод растеризації зрізів воксельних моделей даних, що ґрунтується на використанні ткацьких технологій растеризації. Наразі, необхідно виконати реалізацію відповідного йому алгоритму, дослідити його роботу та здійснити порівняння отриманих результатів із результатами найближчих аналогів. В якості найближчого аналога буде розглянуто точний ткацький алгоритм растеризації зрізів воксельних моделей даних.

Оскільки серед порівнюваних характеристик будуть такі параметри як швидкодія, то, відповідно, достовірність результатів під час порівняння роботи алгоритмів буде залежати значною мірою від програмної реалізації того чи іншого алгоритму. Таким чином, питанню програмної реалізації досліджуваних алгоритмів растеризації слід приділити особливу увагу.

Перш за все, необхідно виокремити загальні вимоги до розроблюваних алгоритмів:

- зменшити кількість обчислень у програмних циклах настільки, наскільки це можливо;
- усюди, де це можливо, використовувати цілочисельну арифметику в обчисленнях;
- надавати перевагу більш простим операціям, уникати складних, порівняно витратних, операцій (наприклад, використовувати операції додавання замість множення);

- уникати використання математичних функцій, замінювати їх використання на використання математичних операторів (наприклад, використовувати операцію множення замість виклику спеціальної функції для піднесення числа до степеня);
- та інші.

Слід зазначити, що на швидкість роботи алгоритму можуть значно впливати використовувані структури даних. Тому необхідно усюди, де використовуються однакові, не специфічні для методу, структури даних використовувати їх однакову реалізацію. Мається на увазі, що для однакових цілей, наприклад, для збереження ланцюжків зсувів координат, у всіх алгоритмах має використовуватися одна й та сама однаково реалізована структура даних.

Як відомо, при реалізації алгоритмів дуже часто виникає ситуація, коли прийом, застосований, наприклад, для підвищення швидкодії, вимагає значного зростання обсягу використовуваної оперативної пам’яті. Тому, для об’єктивного порівняння алгоритмів, при їх реалізації слід враховувати обсяг наданих середовищем ресурсів та відповідні обмеження.

Досліджувані алгоритми мають бути реалізовані у вигляді окремих модулів, що надають доступ до відповідних програмних функцій, які здійснюють растеризацію зрізів воксельної моделі даних заданої розмірності. Вхідними даними мають бути:

- розмірність досліджуваної воксельної моделі (довжина, висота та ширина у вокселях);
- параметри заданої площини зрізу.

Розроблені алгоритми мають бути об’єднані у єдину бібліотеку для подальшого практичного використання.

3.2. Засоби та технології програмної реалізації методу

Вибір мови програмування може мати суттєвий вплив на ефективність роботи алгоритмів. При виборі мови програмування, на якій

будуть реалізовані зазначені вище алгоритми растеризації зрізів слід враховувати цілу низку наступних критеріїв:

- тип реалізації (компільовані та інтерпретовані мови);
- наявність бібліотек із необхідними алгоритмами та структурами даних;
- наближеність семантики мови до машинного коду (мови низького рівня, мови високого рівня);
- підтримка низькорівневих засобів для виділення та управління пам’яттю;
- інші параметри.

Зважаючи на поставлені задачі, для реалізації алгоритмів слід обрати компільовану мову програмування, засоби якої здатні забезпечити швидку обробку великого обсягу даних, і її використання має бути доцільно із практичної точки зору для можливого впровадження розроблених алгоритмів у схожі дослідницькі або комерційні проекти. Тому пропонується оглянути мови C++, C# та Java. Дані мови представляють клас мов високого рівня, на яких можуть бути реалізовані проекти довільної складності, в тому числі і обчислювально-витратні алгоритми.

Усі із вище зазначених технологій пропонують засоби для програмування, які у повній мірі задовольняють вимогам, висунутим поставленою задачею. Перелічені мови програмування мають схожий синтаксис, в тій чи іншій мірі подібний до синтаксису мов сімейства C, та забезпечені сучасними та продуктивними стандартними бібліотеками із необхідними структурами даних. Відповідне програмне забезпечення, яке необхідно створити, може бути реалізовано із мінімальними відмінностями у програмному коді. Тому єдиним важливим критерієм вибору залишається продуктивність виконання програмного коду.

Загальновизнаним фаворитом з точки зору швидкодії та ефективності є мова програмування C++, хоча з іншого боку вона відрізняється від C# та Java дещо більшою складністю кінцевого

програмного коду, і ,відповідно, швидкість розробки програмного забезпечення є меншою [41]. З огляду на специфіку поставленої задачі, останній аспект не є критичним в рамках даної роботи.

Для порівняння ефективності виконання програмного коду в контексті даної роботи було виконано ряд замірів показників швидкодії для різних алгоритмів растеризації прямих на площині. Тестувалася одна і та сама версія алгоритму для растеризації, написана на мовах C++, C# та Java.

Результати тестування подано у табл. 3.1.

Таблиця 3.1

Порівняння швидкодії алгоритмів растеризації прямих
для різних мов програмування

Пряма	Точний ткацький алгоритм	Алгоритм Брезенхема	Алгоритм DDA
C++	0,4323 мс	0,4424 мс	0,4883 мс
C#	0,4596 мс	0,4627 мс	0,5214 мс
Java	0,6312 мс	0,6621 мс	0,7041 мс

Після проведення аналізу оглянутих мов програмування та технологій було вирішено виконати реалізацію алгоритмів растеризації зрізів на мові програмування C++ та розробити відповідне програмне забезпечення для растеризації зрізів тривимірних воксельних моделей даних у вигляді окремої утиліти.

C++ було обрано через високу продуктивність цієї мови та наявність можливості безпосереднього управління пам’яттю. Для C++ існує багато готових стандартних функцій та структур даних, готових до використання, що значно спрощує та пришвидшує розробку програмного забезпечення. На відміну від інших розглянутих високорівневих мов, програми на C++ одразу компілюються у машинні команди і не потребують спеціального середовища виконання, на відміну від віртуальної машини Java або CLR для мов платформи .NET [42, 43]. Цим забезпечується не тільки

збільшення продуктивності виконання програм, але і усунення сторонніх чинників, які можуть вплинути на часові показники роботи алгоритму, наприклад виклик команд для звільнення пам’яті Java або .NET.

3.3. Програмна реалізація алгоритмів

3.3.1. Підготовчий етап

В ході реалізації точного ткацького алгоритму растеризації площин зрізів та запропонованого у даній роботі алгоритму було виявлено, що для обох цих випадків можна виділити, так званий, підготовчий етап растеризації. Дії, що виконуються на даному етапі, є практично однаковими для обох із вище вказаних методів, тому доцільно буде розглянути їх окремо від матеріалу, який безпосередньо описує принципи роботи та особливості їхньої реалізації.

Метою підготовчого етапу растеризації є обробка вхідних даних та виокремлення випадків, для яких не має необхідності застосовувати розглянуті алгоритми у їхніх загальних рисах, але натомість можна використати їх дещо спрощену версію. Як буде показано надалі, спрощення растеризації, а, відповідно, і її пришвидшення, можна досягти оброблюючи різні випадки окремо, в залежності від їх специфіки.

Відповідно до потреб програмної реалізації, для вирішення окремих підзадач було створено набір відповідних функцій, опис яких буде наведено згодом. Згадані функції реалізовано у вигляді методів окремого класу Slicer, який має доступ до розмірностей моделі, зріз якої необхідно отримати.

Вхідними даними для виконання растеризації є параметри моделі, тобто її розмірності (ширина, висота, довжина), а також параметри площини зрізу. Площина зрізу задається загальним рівнянням площини, і, відповідно, вхідними параметрами будуть коефіцієнти *A, B, C* та *D*. Таке подання площини зрізу на практиці є найбільш зручним та зрозумілим.

Для зручності роботи із сутностями, такими як площини, лінії та точки, у розробленій програмі було реалізовано відповідні класи Plane та

Point. У лістингу 3.1 наведено спрощену версію програмного коду для відповідних класів.

Лістинг 3.1. Реалізація класів Point та Plane

```
class Point {
public:
    int x, y, z;
    Point(int x, int y, int z);
}

class Plane {
public:
    Plane(double a, double b, double c, double d);
    double getA() const;
    double getB() const;
    double getC() const;
    double getD() const;
    void setA(double a);
    void setB(double b);
    void setC(double c);
    void setD(double d);
    Point solve(double coord1, double coord2, char variable) const;
private:
    double a, b, c, d;
}
```

На початку підготовчого етапу на основі параметрів площини зрізу визначаються та обчислюються координати вершин многокутника, який представляє собою фігуру зрізу.

Як зазначалося у розділі 2, всього можливо шість варіантів форми зрізу:

- шестикутник;
- п’ятикутник;
- чотирикутник (прямокутник, паралелограм, трапеція);
- трикутник;
- відрізок (одне із ребер моделі);
- точка (одна із вершин моделі).

Для визначення вершин многокутника було розроблено спеціальну функцію getSliceVertices, яка на основі параметрів вхідної площини зрізу обчислює координати вершин. Спочатку знаходяться точки перетину заданої площини зрізу із прямими, яким належать ребра воксельної моделі.

Після знаходження всіх унікальних точок перетину із вказаними прямими, зберігаються лише ті точки, які входять у межі заданої моделі. Фрагмент відповідного коду наведено у лістингу 3.2.

Лістинг 3.2. Пошук та перевірка вершини зрізу

```
Point point = plane.solve(0.0, 0.0, 'x');
if (model->contains(point)) {
    vertices.insert(point);
}
```

Метод solve класу Plane дозволяє знайти точку яка належить площині по двом заданим координатам. Таким чином, після знаходження дванадцяти точок перетину площини із прямими, на яких лежать ребра моделі, залишаються лише ті, які лежать в межах моделі, а отже і є вершинами многокутника зрізу. Випадки, коли площина зрізу паралельна до кількох ребер моделі оброблюються окремо.

Після визначення кількості вершин зрізу воксельної моделі можна здійснити вибір типу подальшої растеризації.

У найбільш тривіальних випадках подальша обробка може не знадобитися взагалі. Так, наприклад, якщо площина зрізу не перетинає воксельну модель тривимірного медичного зображення, то в результаті кількість вокселів, що представляють даний зріз, буде дорівнювати нулю, тобто отримаємо порожню множину вокселів. Інший простий випадок полягає в тому, що площина зрізу дотикається до воксельної моделі лише в одній точці – її вершині. В такому разі множина вокселів зрізу буде складатися лише із однієї точки, яка уже знайдена на етапі пошуку вершин многокутника зрізу воксельної моделі.

В інших випадках, коли кількість вершин многокутника зрізу дорівнює двом, трьом, чотирьом, п’ятьом або шести вершинам, постає необхідність у більш складній растеризації зрізів.

Якщо зріз має дві вершини, то необхідно виконати растеризацію лише одного єдиного ребра воксельної моделі. Ця задача є досить тривіальною з точки зору програмування та обробки воксельних даних.

Для растеризації ребра моделі необхідно визначити його положення відносно самої моделі, тобто дізнатись, яке сама ребро необхідно растеризувати. Знаючи положення ребра, можна виконати його растеризацію, запустивши простий цикл з лічильником в межах від нуля до значення відповідної розмірності. Растеризація в випадку, коли ребро паралельне до осі *Oz*, представлена у фрагменті програмного коду, наведеному у лістингу 3.3.

Лістинг 3.3. Растеризація ребра воксельної моделі

```
if (v1.x == v2.x && v1.y == v2.y) {
    int x = v1.x, y = v1.y;
    for (int z = 0; z < model->getWidth(); ++z) {
        points.push_back(Point(x, y, z));
    }
}
```

В тому випадку, коли зріз воксельної моделі має три вершини, тобто зріз має трикутну форму, якісь специфічні оптимізації для растеризації такого зрізу застосовувати недоцільно. Тому, якщо отриманий многокутник представлений трикутником, можна одразу перейти до растеризації площини зрізу запропонованим в роботі методом або точним ткацьким алгоритмом растеризації. Деталі такої растеризації будуть розглянуті у наступних двох підрозділах.

Якщо зріз має чотири вершини, то в цьому разі необхідно передбачити обробку таких ситуацій, як растеризація ортогональних зрізів, тобто зрізів, які перпендикулярні до однієї із осей *Ox*, *Oy* або *Oz*. У лістингу 3.4 наведено код функції rasterizeOrthogonalX, яка виконує растеризацію зрізу, ортогонального до осі *Ox*.

Аналогічним чином реалізовані функції rasterizeOrthogonalY та rasterizeOrthogonalZ для растеризаці зрізів ортогональних до осей *Oy* та *Oz* відповідно.

Лістинг 3.4. Реалізація функції rasterizeOrthogonalX

```
point_vec rasterizeOrthogonalX(int offset) {
    point_vec points;
    for (int j = 0; j < model->getHeight(); ++j) {
        for (int k = 0; k < model->getWidth(); ++k) {
            points.push_back(Point<int>(offset, j, k));
        }
    }
    return points;
}
```

Також необхідно брати до уваги ситуації, коли чотирикутні зрізи виявляються паралельними до координатних осей. Растеризація зрізів у таких випадках теж не потребує використання яких-небудь спеціальних алгоритмів і є досить тривіальною задачею. Необхідно обрати три точки із чотирьох вершин, які мають утворити два відрізки – master-лінію та base-лінію (див. розділ 2). При цьому master-лінією має бути пряма, паралельна до однієї координатних осей. Растеризацію base-лінії можна виконувати алгоритмом Брезенхема для побудови ліній, і на кожній новій ітерації виконувати растеризацію master-лінії. У лістингу 3.5 продемонстровано фрагмент коду, де представлено таку растеризацію для чотирикутного зрізу, поверхня якого є паралельною до осі *Ox*.

Лістинг 3.5. Растеризація чотирикутного зрізу, паралельного до осі *Ox*

```
drawing::BresenhamsLineAlgorithm(v1.y, v1.z, v2.y, v2.z, [&](int y, int z)
{
    for (int x = 0; x < model->getLength(); ++x) {
        points.push_back(Point(x, y, z));
    }
});
```

Для усіх інших випадків необхідно застосовувати нетривіальні способи растеризації зрізів воксельних моделей, наприклад, точним ткацьким методом растеризації зрізів або методом, запропонованим у даній роботі. Опис алгоритмів, що реалізують вище зазначені методи, буде розглянуто далі.

3.3.2. Реалізація алгоритму запропонованого методу

Запропонований метод растеризації зрізів воксельних моделей даних умовно складається із двох етапів:

- етапу ініціалізації, на якому відбувається растеризація master-лінії, обчислення ланцюжка зсувів та границь фрагментів master-лінії;
- етапу растеризації площини зрізу, під час якого ітеративно будуються фрагменти master-лінії, які разом утворюють поверхню шуканого зрізу воксельної моделі.

У програмній реалізації алгоритму запропонованого методу процес растеризації площини зрізу був поділений на такі підзадачі:

- пошук координат master-лінії та base-лінії із рівняння площини зрізу;
- побудова зрізу, яка об’єднує у собі вище зазначені етапи.

Перед початком растеризації на основі заданого рівняння площини зрізу визначаються master-лінія та base-лінія. У другому розділі було описано сам метод. Згідно раніше зазначених викладок, master-лінія та base-лінія мають задовольняти наступним вимогам:

- лінії мають лежати в ортогональних площинах;
- площини, в яких лежать лінії, мають бути ортогональні до координатних осей;
- кут між master-лінією та площиною в якій лежить base-лінія має бути більшим або дорівнювати 45°.

Для визначення кінців відрізків master-лінії та base-лінії спочатку знаходяться вершини основного трикутника, який перетинає задану воксельну модель і в якому лежить многокутник шуканого зрізу. Вершини такого трикутника лежать на трьох прямих, яким належать ребра воксельної моделі. Координати даних вершин шукаються аналогічно до того, яким чином знаходяться вершини многокутника зрізу у функції

getSliceVertices. Вибір трьох прямих, на яких знаходяться вершини шуканого трикутника, залежить від параметрів заданої площини зрізу воксельної моделі.

Всього можливо вісім варіантів вибору даних прямих, який залежить від напрямку вектора нормалі площини зрізу. Проте якщо усі отримані вершини лежать на координатних осях *Ox*, *Oy* та *Oz* і при цьому мають від'ємні координати, то це свідчить про те, що площина зрізу не перетинає задану воксельну модель. Слід зазначити, що якщо попередньо було виконано перевірки підготовчого етапу, то даний випадок можна окремо не розглядати, оскільки його обробка вже буде виконана на підготовчому етапі.

Описана вище процедура пошуку основного трикутника реалізована функцією defineBaseTriangle. В результаті її використання буде отримано три вершини шуканого трикутника. Після визначення цих трьох точок необхідно визначити, яка із них задовольняє вимогам, висунутим до master-лінії та base-лінії і є початком цих відрізків. Дві інших точки будуть кінцями master-лінії та base-лінії відповідно. Сигнатура даної функції наведена у лістингу 3.6.

Лістинг 3.6. Сигнатура функції defineBaseTriangle

```
void defineBaseTriangle(const Plane& plane,
                        Point& v1,
                        Point& v2,
                        Point& v3);
```

Нараховується всього шість комбінацій master-ліній та base-ліній. Для зручності пропонується прийняти наступні позначення для трьох вершин шуканого трикутника:

- V_1 – вершина, що лежить на прямій, паралельній до *Ox*;
- V_2 – вершина, що лежить на прямій, паралельній до *Oy*;
- V_3 – вершина, що лежить на прямій, паралельній до *Oz*.

На рис. 3.1 зображено основний трикутник $V_1V_2V_3$ та обрані master-лінія (відрізок V_2V_1') та base-лінія (відрізок V_2V_3').

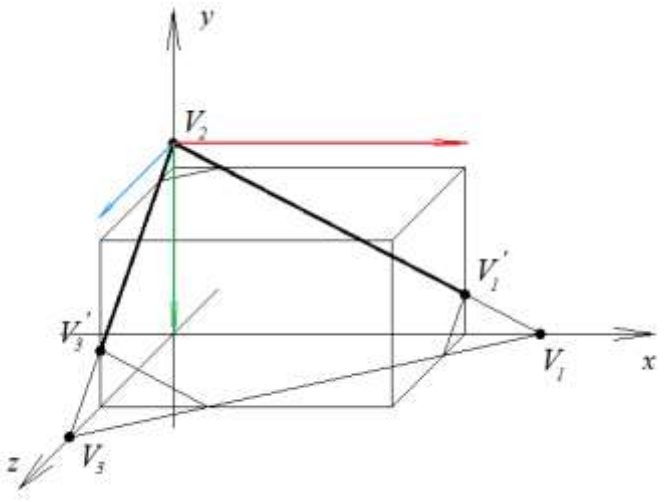


Рис. 3.1. Вершини основного трикутника для шестикутного зрізу

Для ефективної растеризації необхідно заздалегідь визначитися із напрямками, у якому будуть змінюватися відповідні координати вокселів. На рис. 3.1 такі напрямки позначені кольоровими стрілками, початок яких розташований в вершині основного трикутника V_2.

В залежності від того, як будуть змінюватися координати растеризованих точок, потрібно обирати функцію, яка здійснює таку растеризацію. Як показало практичне використання розробленого алгоритму, застосування таких спеціалізованих функції є набагато ефективнішим, ніж використання більш універсальних, проте менш продуктивних функцій растеризації.

Результатом виконаних операцій, що описані вище, будуть коректно визначені master-лінія та base-лінія, а також визначено напрямки зміни координат вокселів. Ця інформація буде використана в подальшому при растеризації зрізу. Слід зазначити, що точки початку обох ліній співпадають.

Після визначення master-лінії та base-лінії виконується етап ініціалізації. Для кожного конкретного випадку растеризація ліній, а, відповідно, і площини зрізу будуть відрізнятися відповідно до того, як

зорієнтовані master-лінії та base-лінії одне відносно одного. Припустимо, що master-лінія лежить у площині *Oxy*, а base-лінія – у площині *Oyz*. Тоді растеризація master-лінії буде одержана у результаті виконання наступного фрагменту коду, представленого у лістингу 3.7.

Лістинг 3.7. Растеризація master-лінії

```
int prevY = -1;
int indexL, indexR;
std::vector<int> C, L, R;

drawing::BresenhamsLineAlgorithm(x1, y1, x2, y2, [&](int x, int y) {
    if (prevY != y) {
        C.push_back(1);
        L.push_back(x);
        prevY = y;
    } else {
        C.push_back(0);
    }
});
C[0] = 0;
```

Таким чином, растеризація master-лінії виконується за допомогою алгоритму Брезенхема для побудови ліній у двовимірному просторі. Початок та кінець master-лінії задаються координатами (x_1, y_1) та (x_2, y_2) відповідно.

Процес растеризації зводиться до визначення ланцюжка зсувів для координати *y*. Одночасно із цим визначаються ліві границі фрагментів master-лінії. Визначення правих границь фрагментів master-лінії наведено у лістингу 3.8.

Лістинг 3.8. Знаходження правих границь фрагментів master-лінії

```
for (size_t i = 1; i < L.size(); ++i) {
    R.push_back(L[i] - 1);
}
R.push_back(L.back());
```

Далі встановлюються індекси, що вказують на початкові значення границь фрагментів master-лінії (лістинг 3.9).

Лістинг 3.9. Встановлення початкових індексів для границь

```
indexL = (y1 < height) ? 0 : y1 - (height - 1);
indexR = R.size() - 1;
```

На цьому етап ініціалізації завершено. Тепер можна переходити до реалізації основного етапу методу в алгоритмі.

Основний етап полягає у ще одному виконанні алгоритму Брезенхема, проте цього разу вже для base-лінії. На кожному кроці виконання даного алгоритму буде растеризовано один фрагмент master-лінії у межах, визначених на етапі ініціалізації. В результаті виконання усіх ітерацій алгоритму Брезенхема буде растеризовано площину зрізу заданої воксельної моделі.

Нехай початок та кінець base-лінії задано координатами (y_1, z_1) та (y_2, z_2) відповідно. Тоді растеризація площини зрізу виконується наступним чином, як показано в лістингу 3.10.

Лістинг 3.10. Основний етап растеризації

```
prevY = y1;

drawing::BresenhamsLineAlgorithm(y1, z1, y2, z2, [&](int y, int z) {
    int y0 = (y < height) ? y : height;

    if (prevY != y) {
        if (indexL != 0) indexL--;
        if (y < R.size()) indexR--;
        prevY = y;
    }
    for (int x = L[indexL]; x <= R[indexR]; ++x) {
        y0 -= C[x];
        points.push_back(Point(x, y0, z));
    }
});
```

На кожному новому кроці алгоритму Брезенхема змінюються границі фрагментів master-лінії, а потім растеризується поточний фрагмент у простому циклі з лічильником у вказаних межах.

У лістингу 3.11 наведено код функції, що реалізує алгоритм Брезенхема для побудови ліній, який було використано для растеризації master-лінії та base-лінії.

Дана реалізація алгоритму Брезенхема працює для двох довільно заданих точок, при цьому такі випадки як растеризація горизонтальних або вертикальних прямих, а також прямих із кутом нахилу у 45° розглядаються

окремо, що дозволяє зменшити кількість операцій у найбільш тривіальних ситуаціях.

Лістинг 3.11. Реалізація алгоритму Брезенхема для побудови ліній

```
void BresenhamsLineAlgorithm(int x1, int y1, int x2, int y2,
                             std::function<void(int, int)> callback) {
    int deltaX = abs(x2 - x1);
    int deltaY = abs(y2 - y1);
    int signX = x1 < x2 ? 1 : -1;
    int signY = y1 < y2 ? 1 : -1;
    int x, y;
    int error = 0;

    if (deltaY == 0) {
        for (x = x1, y = y1; x != x2; x += signX) callback(x, y);
    } else if (deltaX == 0) {
        for (x = x1, y = y1; y != y2; y += signY) callback(x, y);
    } else if (deltaY < deltaX) {
        for (x = x1, y = y1; x != x2; x += signX) {
            callback(x, y);
            error += deltaY;
            if (error * 2 >= deltaX) {
                y += signY;
                error -= deltaX;
            }
        }
    } else if (deltaX < deltaY) {
        for (x = x1, y = y1; y != y2; y += signY) {
            callback(x, y);
            error += deltaX;
            if (error * 2 >= deltaY) {
                x += signX;
                error -= deltaY;
            }
        }
    } else {
        for (x = x1, y = y1; x != x2; x += signX, y += signY)
callback(x,y);
    }

    callback(x, y);
}
```

Також слід зазначити, що усі наведені вище фрагменти коду є спрощеними для наочності викладення та не представляють собою кінцеву максимально оптимізовану версію.

3.3.3. Реалізація точного ткацького алгоритму растеризації

Точний ткацький алгоритм растеризації зрізів воксельних моделей даних описано у роботах попередніх дослідників. Було викладено теоретичний матеріал, де доведено ряд фактів стосовно растеризації прямих, заданих рівнянням із кутовим коефіцієнтом. Дослідження

показали, що ланцюжок зсувів (див. формулу 2.3) для однієї із координат може повторюватись із певною періодичністю. Відповідно до встановленого факту, потреба у обчисленні усіх елементів такого ланцюжка відпадає. Достатньо лише обчислити фрагмент ланцюжка, щоб потім його просто копіювати із певним зсувом. Дана можливість дозволяє значно пришвидшити растеризацію окремих ліній. Для порівняння реалізацій алгоритмів растеризації зрізів, у лістингу 3.12 наведено код для обчислення ланцюжка зсувів [33].

Лістинг 3.12. Обчислення ланцюжка зсувів в точному ткацькому алгоритмі

```
int computeChain(int c[], int p, int q, int k) {
    int pInverse, d = 2 * p - q, inc0 = 2 * p, inc1 = 2 * (p - q);
    int q1 = q + 1;
    int qk = q + k;

    for (int i = 1; i < q1; ++i) {
        if (d <= 0) {
            d += inc0;
            c[i] = 0;
        } else {
            d += inc1;
            c[i] = 1;
        }

        if (((p * i) % q) == 1) pInverse = i;
    }
    c[0] = c[q];

    for (int i = 1; q * i < qk; ++i)
        memcpy(&c[q * i], &c[0], q * sizeof(c[0]));

    return pInverse;
}
```

Варто відмітити, що у лістингу 3.12 наведено приклад використання низькорівневої функції для швидкого копіювання ділянки пам’яті – memcpy.

У лістингу 3.13 представлено код повного виконання растеризації прямокутної ділянки площини точним ткацьким алгоритмом растеризації [33].

Лістинг 3.13. Растеризація площини точним ткацьким алгоритмом

```
std::vector<Point> plane3D(int A, int B, int C, int D,
        int x0, int x1, int y0, int y1, int k) {
    std::vector<Point> points;

    int z, s, p, q, g, pInvers;
    int d, f, inc0 = 2 * B, inc1 = 2 * (B - C);
    int* c;

    g = math::gcd(A, C);
    p = A / g;
    q = C / g;
    c = new int[k + q];

    pInvers = computeChain(c, p, q, k);
    z = (int) round((A*x0 + B*(y0 - 1) + D) / (float) C);
    d = 2 * (A*x0 + B*y0 - C*z + D) - C;
    f = (q % 2) ? 3*C : 3*C + g;

    for (int y = y0; y <= y1; ++y) {
        s = (pInvers * (int) round((d + f)/(2.0f * g))) % q;
        if (d <= 0){
            d += inc0;
        } else {
            d += inc1;
            z++;
        }

        int _z = z;
        points.push_back(Point(x0, y, _z));

        for (int _x = x0 + 1; _x <= x1; ++_x) {
            _z += c[++s];
            points.push_back(Point(_x, y, _z));
        }
    }

    delete[] c;
    return points;
}
```

В даному випадку наведена спрощена реалізація алгоритму, яка поширюється на випадки растеризації площин, заданих рівнянням $Ax + By - Cz + D = 0$, де відповідні коефіцієнти A, B, C мають не дорівнювати 0, при чому $0 < A,\ B \le C$.

3.4. Висновки

В ході розробки прорамного забезпечення, яке реалізує алгоритм запропонованого у роботі методу, а також точний ткацький алгоритм растеризації, було складено порівняльну характеристику найбільш підходящих інструментаріїв для реалізації. В результаті такого порівняння

було прийнято рішення виконати вище зазначені алгоритми на мові програмування C++.

Під час проведеного дослідження були висунуті вимоги до реалізації розроблюваних алгоритмів. Ключові моменти у роботі алгоритмів розглянуті найбільш детально, було наведено відповідні фрагменти програмного коду.

Обидві реалізації алгоритмів були виконані із використанням максимально простих і, водночас, ефективних конструкцій програмного коду з метою можливості проведення подальшого об’єктивного тестування швидкодії алгоритмів та їх порівняння.

4. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Результати тестування

Реалізовані точний ткацький алгоритм растеризації, а також алгоритм, що реалізує запропонований у роботі метод були відповідним чином протестовані на різних наборах вхідних даних. Найбільш ретельно проводилося тестування швидкодії алгоритмів, оскільки основною метою даної роботи була розробка більш ефективного методу растаризації, що дозволяє прискорити растеризацію зрізів воксельних моделей даних, в тому числі, тривимірних медичних зображень.

Робота реалізованих алгоритмів перевірялася на моделі розмірністю 256x256x256 вокселів. Обидва алгоритми були виконані на мові програмування C++. Частини програм, що відповідали за етапи ініціалізації алгоритмів тестувалися окремо. Також окремо тестувалася швидкодія растеризації відрізків прямих. Усі тести для обох алгоритмів проводилися в однакових умовах на однакових наборах даних на одному і тому ж комп'ютері.

4.1.1. Тестування швидкодії растеризації відрізків прямих

Особливість ткацьких методів растеризації поверхонь полягає в тому, що растеризується лише одна крива (пряма) у двовимірному просторі, яка потім растеризується уздовж іншої кривої (прямої) із певним зсувом. Тому, досліджуючи швидкодію таких методів, необхідно приділити увагу растеризації кривих (прямих) у двовимірному просторі.

У табл. 4.1 представлено результати порівняння швидкодії растеризації відрізків прямих точним ткацьким алгоритмом растеризації та алгоритмом Брезенхема для побудови ліній. Також для порівняння було розглянуто алгоритм DDA [44] для побудови ліній. З даних таблиці видно, що прямі задані рівняннями із кутовим коефіцієнтом. Значення кутових коефіцієнтів обиралися з метою протестувати різні випадки у роботі алгоритмів. Швидкість роботи алгоритмів Брезенхема та DDA не залежить

суттєво від вхідних даних, на відміну від точного ткацького алгоритму растеризації, де від нахилу прямої залежить довжина періоду ланцюжка зсувів, а отже і швидкість растеризації.

Таблиця 4.1

Порівняння швидкодії алгоритмів растеризації прямих

Пряма	Довжина прямої	Точний ткацький алгоритм	Алгоритм Брезенхема	Алгоритм DDA
$y = \frac{255}{256}x$	4096	0,6396 мс	0,6723 мс	0,7005 мс
$y = \frac{1}{2}x$	4096	0,6692 мс	0,6692 мс	0,7 мс
$y = \frac{1}{10}x$	4096	0,6396 мс	0,6676 мс	0,7004 мс
$y = \frac{7}{256}x$	4096	0,6443 мс	0,6656 мс	0,7067 мс
$y = \frac{1365}{4096}x$	4096	0,6864 мс	0,6771 мс	0,7223 мс
$y = \frac{1}{4}x$	4096	0,6849 мс	0,6879 мс	0,7203 мс
$y = \frac{819}{4096}x$	4096	0,695 мс	0,6817 мс	0,7132 мс
$y = \frac{341}{2048}x$	4096	6,786 мс	0,6802 мс	0,7222 мс

З отриманих результатів можна зробити висновок, що жоден із розглянутих алгоритмів не може забезпечити суттєвої та стабільної переваги порівняно із аналогами. Алгоритм DDA показав найгірші результати. Растеризація прямих точним ткацьким алгоритмом растеризації в деяких випадках виконувалася ефективніше за растеризацію алгоритмом Брезенхема. Це зумовлено особливістю даного алгоритму, оскільки враховується періодичність з якою повторюються фрагменти ліній і відбувається їх копіювання. Проте в більшості ситуацій алгоритм Брезенхема спрацьовував швидше за конкурентів.

4.1.2. Тестування швидкодії растеризації зрізів

Тестування швидкодії растеризації зрізів полягає у замірі часових показників роботи алгоритмів від моменту передачі вхідних даних до моменту отримання результату. Таким чином, відбувалося тестування швидкодії растеризації зрізу уцілому. Крім того, етапи ініціалізації тестувалися окремо. У табл. 4.2 та табл. 4.3. наведено результати замірів швидкодії растеризації зрізів.

Таблиця 4.2

Результати порівняння швидкодії алгоритмів растеризації зрізів (етап ініціалізації)

Площина	Точний ткацький алгоритм растеризації		Алгоритм запропонованого методу	
	Кількість вокселів	Швидкодія (мс)	Кількість вокселів	Швидкодія (мс)
$x+y+z=0$	32896	0,0157 мс	32896	0,224 мс
$x+2y+2z=0$	49152	0,0186 мс	49152	0,2687 мс
$3x+2y+5z=0$	65536	0,0175 мс	65536	0,2529 мс
$10x+7y+8z=0$	36992	0,0162 мс	36992	0,235 мс
$4x+2y+3z-10=0$	44609	0,0176 мс	44528	0,2531 мс
$48x+42y+64z-100=0$	54916	0,0193 мс	54801	0,2801 мс
$37x+30y+32z-57=0$	35386	0,016 мс	35450	0,2313 мс
$99x+101y+128z+80=0$	48463	0,0184 мс	48444	0,2643 мс
$x+y+z-150=0$	48646	0,0185мс	48646	0,2678 мс
$34x+32y+32z+15=0$	32896	0,0159 мс	32896	0,225 мс

Алгоритм запропонованого методу використовує для растеризації master-лінії та base-лінії алгоритм Брезенхема, реалізація якого була приведена вище (див. лістинг 3.2). Із представлених результатів очевидно, що етап ініціалізації займає набагато менше часу для точного ткацького алгоритму растеризації зрізів. Натомість, алгоритм запропонованого методу растеризації зрізів надає виграш у швидкодії під час виконання основного етапу.

Таблиця 4.3

Результати порівняння швидкодії алгоритмів растеризації зрізів (основний етап)

Площина	Точний ткацький алгоритм растеризації		Алгоритм запропонованого методу	
	Кількість вокселів	Швидкодія (мс)	Кількість вокселів	Швидкодія (мс)
$x+y+z=0$	32896	7,828 мс	32896	7,2544 мс
$x+2y+2z=0$	49152	9,261 мс	49152	8,6976 мс
$3x+2y+5z=0$	65536	9,7492 мс	65536	9,1816 мс
$10x+7y+8z=0$	36992	8,112 мс	36992	7,5858 мс
$4x+2y+3z-10=0$	44609	8,8302 мс	44528	8,195 мс
$48x+42y+64z-100=0$	54916	9,6799 мс	54801	9,1576 мс
$37x+30y+32z-57=0$	35386	7,915 мс	35450	7,4822 мс
$99x+101y+128z+80=0$	48463	9,1832 мс	48444	8,5517 мс
$x+y+z-150=0$	48646	9,2102 мс	48646	8,6983 мс
$34x+32y+32z+15=0$	32896	7,976 мс	32896	7,293 мс

4.1.3. Тестування точності растеризації зрізів

Тестування точності растеризації зрізів полягало у візуальному співставленні результатів растеризації та перевірці множин отриманих вокселів зрізів на ідентичність. В якості еталону точності було взято реалізований точний ткацький алгоритм растеризації.

Підхід, застосований у роботі точного ткацького алгоритму, полягає у обчисленні координат вокселів із використанням рівняння прямої із кутовим коефіцієнтом. Наприклад, для знаходження координат *x* та *y* в площині *Oxy* виконується обчислення ланцюжка зсувів за формулами (2.1), (2.2) та (2.3). Такий підхід має забезпечувати максимально можливу точність.

Натомість, алгоритм, що реалізує запропонований у роботі метод, використовує алгоритм Брезенхема для растеризації ліній. Алгоритм Брезенхема не передбачає точного обчислення координат [45]. Замість цього використовується спеціальна змінна, в якій зберігається та

накопичується похибка відхилення поточних координат від ідельної прямої. Коли значення змінної почне перевищувати встановлений поріг, значення координат буде відповідним чином скореговано.

Множина вокселів із більшою кількістю елементів, отримана після виконання растеризації одним із методів, зберігалася у спеціальній структурі даних, що забезпечує унікальність кожного елементу в колекції. В реалізованому програмному забезпеченні такою структурою було обрано unordered_set із бібліотеки STL для C++. Далі виконувався пошук вокселів із множини, отриманої іншим методом, в утвореній колекції по ключу. В якості унікального ключа було використано рядок, в якому вказано координати конкретного вокселя. Таким чином була виконана перевірка на входження одного вокселя у різні множини, що належали різним зрізам. Після порівняння множин, до кількості неспівпадінь додавалась різниця між кількостями елементів порівнюваних множин.

На рис. 4.1 зображено зріз тривимірного медичного зображення для площини $x+2y+2z=0$, отриманого точним ткацьким алгоритмом растеризації зрізів.

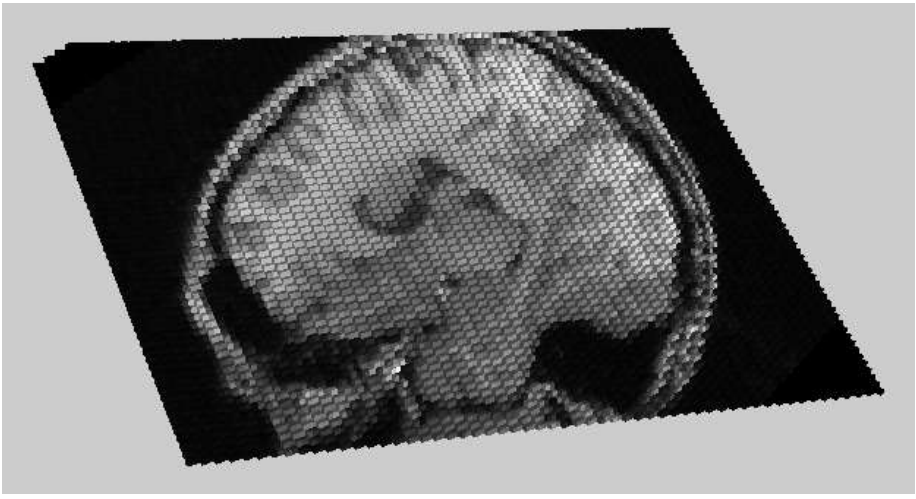


Рис. 4.1. Растеризація зрізу, отриманого
точним ткацьким алгоритмом

На рис. 4.2 зображено зріз тривимірного медичного зображення для площини $x+2y+2z=0$, отриманого алгоритмом, що реалізує запропонований у роботі метод.

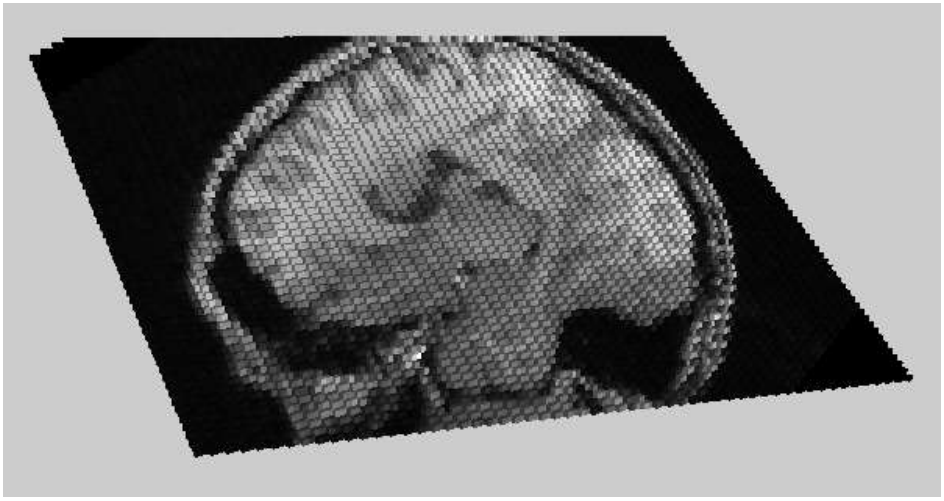


Рис. 4.2. Растеризація зрізу, отриманого
запропонованим алгоритмом

Із наведених вище результатів можна зробити висновок, що точність, яку надає запропонований у роботі метод, візуально не поступається точності точного ткацького алгоритму растеризації зрізів, особливо для тривимірних зображень високої роздільної здатності.

У табл. 4.4 наведено результати порівняння множин вокселів, отриманих при растеризації зрізів розглянутими вище методами.

Таблиця 4.4

Результати порівняння точності алгоритмів растеризації зрізів

Площина	Кількість вокселів		
	Точний ткацький алгоритм растеризації	Алгоритм запропонованого методу	Кількість неспівпадінь
$x+y+z=0$	32896	32896	0
$x+2y+2z=0$	49152	49152	0
$3x+2y+5z=0$	65536	65536	15606
$10x+7y+8z=0$	36992	36992	0
$4x+2y+3z-10=0$	44609	44528	14867
$48x+42y+64z-100=0$	54916	54801	18509
$37x+30y+32z-57=0$	35386	35450	8768
$99x+101y+128z+80=0$	48463	48444	14324
$x+y+z-150=0$	48646	48646	0
$34x+32y+32z+15=0$	32896	32896	0

Як свідчать наведені вище результати растеризації зрізів, помітно, що приблизно у половині випадків растеризація запропонованим у роботі алгоритмом жодним чином не відрізняється від растеризації точним ткацьким алгоритмом. В інших випадках можна спостерігати розбіжності у результатах. Кількість не співпадаючих вокселів може досягати 30%, проте візуально отримані зрізи не будуть значно відрізнятися одне від одного.

4.2. Оцінка запропонованого методу растеризації

Як можна побачити із одержаних результатів, алгоритм, що реалізує запропонований у роботі метод растеризації зрізів воксельних моделей даних, може працювати швидше за точний ткацький алгоритм растеризації.

В середньому перевага по швидкодії може становити від 3 до 5 відсотків. Такий приріст зумовлено спрощенням циклу растеризації фрагментів master-лінії за рахунок визначення границь кожного фрагменту на етапі ініціалізації.

Слід зазначити, що при цьому для запропонованого алгоритму необхідно більше оперативної пам’яті, оскільки обчислені границі фрагментів необхідно окремо зберігати, проте для сучасного апаратного забезпечення виділення необхідного обсягу пам’яті для роботи даного алгоритму не є суттєвою проблемою.

Із результатів тестування було помічено, що етап ініціалізації для точного ткацького алгоритму растеризації зрізів виконується набагато швидше, ніж ініціалізація алгоритму для запропонованого методу. Це зумовлено можливістю використання низькорівневих функцій для роботи із пам’яттю для копіювання фрагментів ланцюжка зсувів. На моделях невеликої розмірності це може забезпечити суттєву перевагу точного ткацького алгоритму растеризації у швидкодії. Проте у більшості випадків, він поступається алгоритму для запропонованого методу, коли мова йде про моделі більшої розмірності. Також може виникнути ситуація, коли ініціалізація для точного ткацького алгоритму растеризації може

виявитися повільнішою через те, що період повтору елементів ланцюжка зсувів може бути достатньо великим, і перевага, яка могла б бути отримана копіюванням ділянок пам’яті, не може бути використана.

При тестуванні точності растеризації було встановлено, що приблизно в половині випадків растеризовані поверхні зрізів майже не відрізняються одне від одного. Візуально ж, відмінності, що присутні на отриманих зрізах практично не помітні для людського ока. Це робить використання запропонованого у роботі методу цілком придатним для практичного використання при вирішенні більшості прикладних задач.

Наостанок необхідно зазначити, що у ході тестування було виявлено певні недоліки у роботі ткацьких алгоритмів загалом. Так, при растеризації зрізів, що розташовані дуже близько до граней воксельної моделі та перетинають їх під достатньо гострим кутом, на лінії перетину спостерігається ефект терас. На рис. 4.3 зображено приклад такого зрізу. Для наочності границі моделі позначені червоним.

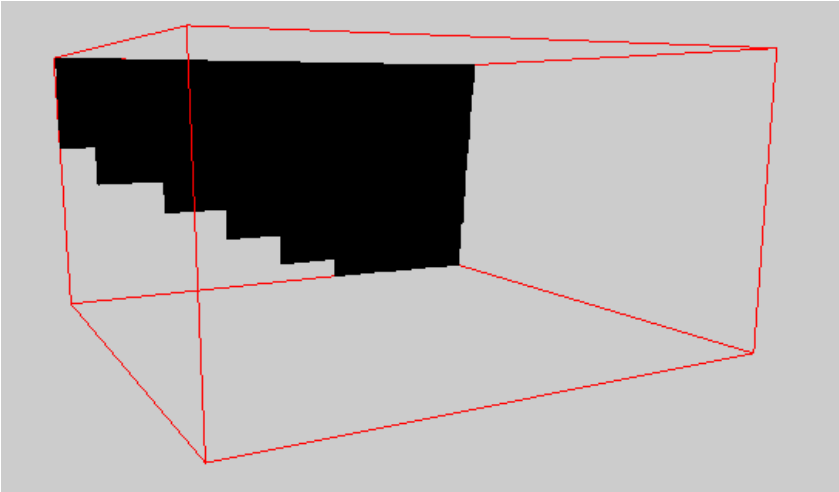


Рис. 4.3. Растеризація зрізу, виконаного під гострим кутом до однієї із граней моделі

На рис. 4.3 чітко видно, що лінія перетину дальньої грані моделі та растеризованого зрізу представлена східчастою ламаною. Дане спотворення зумовлене природою ткацьких алгоритмів растеризації та растеризованих ліній зокрема. Обробку таких випадків необхідно здійснювати окремо.

4.3. Висновки

В ході аналізу результатів тестування та показників швидкодії було встановлено, що запропонований метод може забезпечити прискорення растеризації зрізів воксельних моделей даних великого обсягу.

В порівняння із точним ткацьким алгоритмом растеризації зрізів, алгоритм для запропонованого методу, в загальному випадку, працюватиме швидше за аналог на 3-5%.

Хоча при цьому різниця у часі є незначною, растеризація набору зрізів для воксельних моделей даних великої розмірності, зокрема тривимірних медичних зображень, може бути значно прискорена.

5. ПОБУДОВА БІЗНЕС-МОДЕЛІ

5.1. Опис проблеми

Тривимірне моделювання медичних зображень відіграє велику роль у галузі медицини. Такі зображення отримують, використовуючи спеціальне апаратне забезпечення, шляхом поєднання серії двовимірних зображень під час сканування об’єкту, що досліджується. Візуалізація тривимірних зображень здійснюється спеціалізованим прикладним програмним забезпеченням.

Основною проблемою даної предметної галузі є складність вивчення та дослідження тривимірних медичних зображень через недосконаліть вище згаданого програмного забезпечення та специфіку самої галузі.

Причинами виникнення даної проблеми є:

- обмеженість при перегляді досліджуваної моделі у трьох ортогональних площинах – існуюче альтернативне програмне забезпечення дозволяє вивчати зрізи ортогональні або паралельні площині сканування об’єкту, проте рідко надає змогу зорієнтувати площину зрізу під кутом, необхідним спостерігачеві, що обмежує можливості детального вивчення об’єкту;
- недостовірність та недостатня наочність отриманих результатів – обмеженість перегляду медичних зображень ускладнює діагностику та виявлення подробиць і ключових деталей у стані досліджуваного об’єкта, що призводить до недостатнього розуміння ситуації у цілому та хибних висновків;
- великий об’єм вхідних даних – тривимірні медичні зображення характеризуються великим обсягом даних, які необхідні для їх представлення, що є причиною високих витрат по швидкодії та неефективної візуалізації;

– складність та жорстка спеціалізація існуючих алгоритмів обробки тривимірних зображень – методи та алгоритми обробки тривимірних моделей досить витратні по швидкодії та здебільшого орієнтовані на використання спеціального апаратного забезпечення [33];
– використання спеціалізованого програмного та апаратного забезпечення для візуалізації – через великих обсяг даних, в рамках оптимізації, використовується апаратне та відповідне програмне забезпечення, що підвищує вартість рішення та його доступність;
– небажанні спотворення при використанні GPU – деякі альтернативні рішення дозволяють достатньо ефективно виконувати візуалізацію зрізів тривимірних медичних зображень, використовуючи можливості графічних процесорів, однак це відбувається за рахунок точності та якості візуалізації [46];
– непридатність тривимірних моделей для обробки на CPU – центральні процесори звичайних персональних комп'ютерів не здатні ефективно оброблювати великі обсяги даних тривимірних медичних зображень;
– недостатня швидкодія та точність візуалізації – при візуалізації зрізів тривимірних зображень під довільними кутами зрізу вкрай важко досягти достатнього рівня ефективності при належній точності.

Складність дослідження тривимірних медичних зображень, у свою чергу, породжує наступний ряд проблем:

– обмежена доступність та оперативність діагностики – використання спеціалізованого апаратного забезпечення

призводить до обмеженого доступу для візуалізації тривимірних медичних зображень;

- велика кількість діагностик в процесі обстеження – через недостовірність отриманої інформації у ході діагностики може виникнути потреба у проведенні повторного або додаткового обстеження пацієнтів;
- ускладнене використання у навчальних цілях – через обмежену доступність можливості по застосуванню для навчання медичних фахівців є обмеженими;
- велика кількість експериментів у ході досліджень – при неможливості візуалізації довільних зрізів моделі виникає необхідність у проведенні додаткових експериментів та витрат на дослідження.

Вище перелічені проблеми, що виникають при дослідженні тривимірних медичних зображень представлені на рис. 5.1 у вигляді дерева проблем.

Розробка програмного забезпечення з використанням сучасних технологій та підходів, а також застосування розроблених методів та алгоритмів растеризації зрізів тривимірних зображень усуне ряд проблем, основними з яких є:

- використання спеціального апаратного забезпечення для прискорення растеризації та візуалізації тривимірного зображення;
- недостатня точність та деталізація тривимірного зображення.

Вирішення цих проблем дозволить отримати доступне програмне забезпечення для медичних фахівців, що оперативно надає достовірні результати.

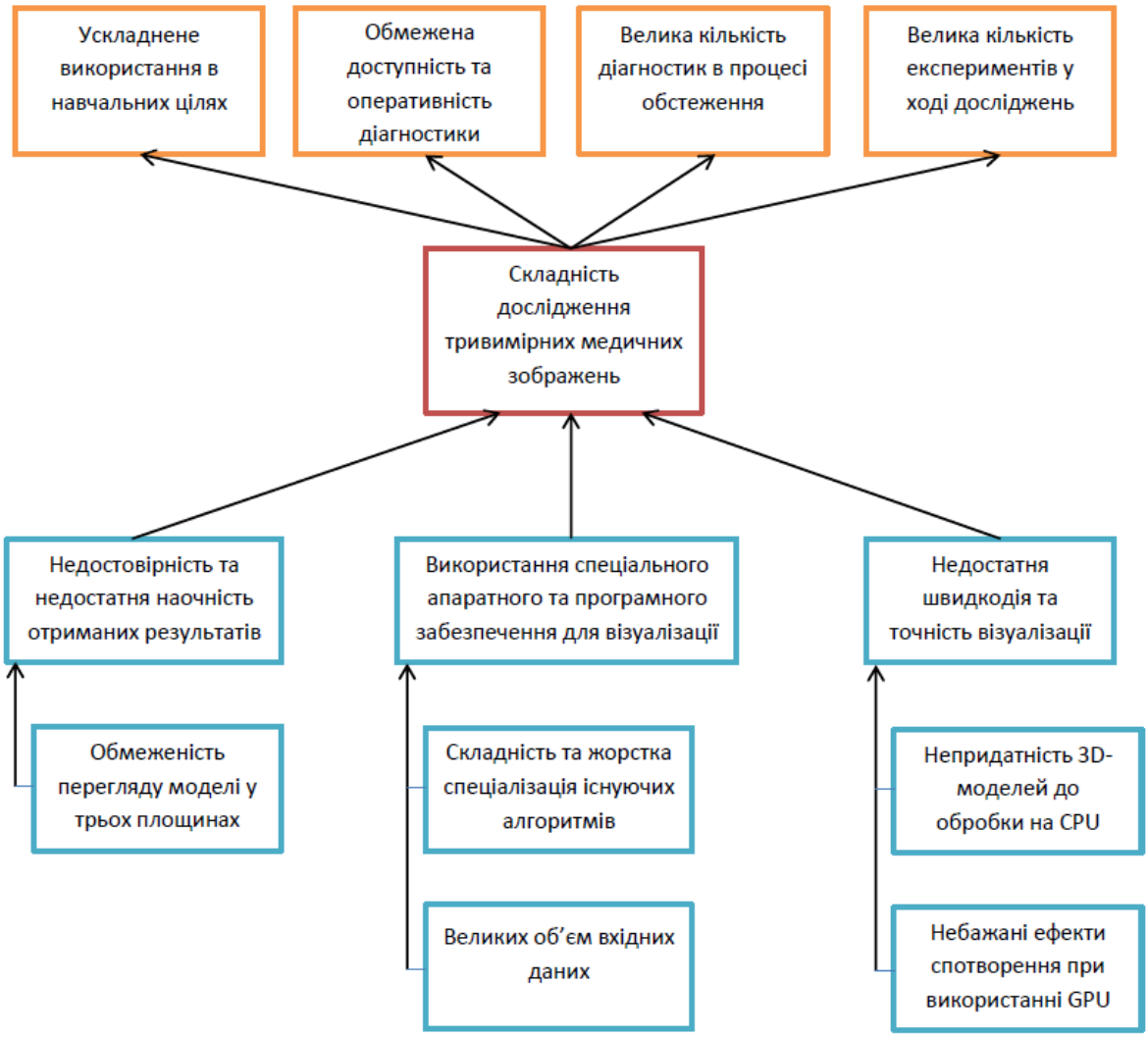


Рис. 5.1. Дерево проблем

5.2. Зацікавлені сторони

Зацікавленими у розробці програмного забезпечення для візуалізації тривимірних медичних зображень можуть бути безпосередньо медичні та суміжні установи.

У вирішенні проблем дослідження та аналізу тривимірних зображень зацікавленні насамперед структури та організації, які безпосередньо працюють із такими зображеннями, проводять діагностику та дослідження із використанням технологій комп'ютерної томографії, магнітно-резонансної томографії, ультразвукової діагностики тощо, оброблюють отримані тривимірні медичні зображення.

У ході аналізу предметної галузі, були виявлені наступні групи зацікавлених сторін:

- державні та приватні медичні заклади;
- науково-дослідні лабораторії, інститути та організації;
- навчальні заклади медичного спрямування.

До цього переліку слід додати потенційних інвесторів та виробників апаратного забезпечення, яке дозволяє отримувати тривимірні зображення (виробники комп'ютерних томографів, засобів ультразвукової діагностики та ін.).

Хоча останні дві групи і не працюють безпосередньо з отриманими зображеннями, проте вони теж можуть бути зацікавлені у розробці відповідного продукту та мають великий вплив.

У табл. 5.1 зведено усі групи зацікавлених сторін, їх інтереси, вплив та стратегії їх приваблення.

Таблиця 5.1.

Зацікавлені сторони

Зацікавлені сторони	Інтереси	Вплив	Стратегії приваблення зацікавлених сторін
Державні та приватні медичні заклади.	Своєчасна та точна діагностика в ході обслуговування пацієнтів. Доступність та відсутність потреби у додатковому апаратному забезпеченні.	Великий	Прийняття участі у тендерах. Презентація власного продукту, демонстрація функціональних можливостей.
Науково-дослідні лабораторії, інститути, організації.	Зменшення кількості експериментів та досліджень. Висока наочність, точність та якість візуалізації.	Середній	Надання можливості тимчасового пробного використання. Забезпечення технічної підтримки та навчання.

Продовження табл. 5.1.

Зацікавлені сторони	Інтереси	Вплив	Стратегії приваблення зацікавлених сторін
Навчальні заклади медичного спрямування.	Можливість використання загальнодоступного програмного забезпечення з навчальною метою на реальних моделях. Підвищення якості навчання медичних працівників.	Низький	Прийняття участі у тендерах. Презентація власного продукту, демонстрація функціональних можливостей. Надання можливості тимчасового пробного використання. Забезпечення технічної підтримки та навчання.
Виробники медичного апаратного забезпечення	Інтеграція програмного забезпечення для візуалізації тривимірних зображень у свої розробки.	Великий	
Інвестори	Вкладення фінансових засобів в розробку з метою отримання прибутку в перспективі.	Великий	

5.3. Комерційне рішення. Основні характеристики

Відповідно до вище зазначених проблем, можна описати кінцевий продукт, що має їх вирішувати.

Передбачається, що кінцевий продукт буде представляти собою програмне забезпечення для персонального комп’ютера і буде застосовуватися для візуалізації тривимірних медичних зображень.

Дане програмне забезпечення є незалежним від апаратного забезпечення виробника та може використовуватись для роботи без прив’язки до нього.

Запропоноване програмне забезпечення буде виконувати поставлені задачі, використовуючи технології апаратного прискорення, можливості графічних процесорів для обчислювальних задач, технології

розпаралелювання програмних процесів, а також розроблених методів та алгоритмів растеризації, що дозволяють ефективно та точно будувати воксельні моделі довільних зрізів тривимірних зображень [3].

Окрім вище зазначених можливостей, кінцевий продукт повинен мати наступні характеристики:

- розроблене програмне забезпечення має працювати на звичайних користувацьких комп'ютерах із середніми технічними параметрами або вище:
  - обсяг оперативної пам'яті – не менше 4Гб;
  - обсяг зайнятої пам'яті жорсткого диску – 1Гб,
  - пам'ять відеокарти – 2Гб;
  - тактова частота процесора – 2,2ГГц.
- повинні підтримуватись поширені файлові формати для збереження тривимірних зображень (NIFTI, DICOM тощо);
- має підтримуватись візуалізація тривимірних зображень обсягом до 1Гб;
- розроблене програмне забезпечення має працювати на сучасних поширених операційних системах x86 та x64 архітектур (Windows 7 та вище, Mac OS, Linux).

5.4. Конкурентні переваги

Серед переваг представленого продукту є простота у використанні, висока якість візуалізації, високий рівень швидкодії, доступність та відносно невисока вартість у порівнянні із сучасними аналогічними рішеннями.

Запропонований продукт забезпечуватиме підтримку широкого набору необхідних функціональних можливостей, що дозволить використовувати його для вирішення досить великого спектру задач:

- діагностики пацієнтів;
- дослідження отриманих тривимірних медичних зображень;

– демонстрацій результатів досліджень;
– використання у навчанні майбутніх медичних фахівців;
– та ін.

Слід зазначити, що використання власних методів та алгоритмів растеризації можна розглядати як основну конкурентну перевагу серед відомих аналогів, оскільки їх використання дозволяє відмовитись від відомих рішень, які, для ефективної роботи, потребують спеціального апаратного забезпечення або ж мають недостатню швидкодію або точність.

5.5. Клієнти та сегменти ринку споживання

З огляду на встановлені проблеми та результати аналізу зацікавлених сторін можна зробити висновок, що запропоноване програмне забезпечення не представляє інтересу для звичайних кінцевих споживачів, які будуть використовувати продукт в рамках своєї професійної діяльності у структурах вищого рівня. Тому кінцеві споживачі (наукові працівники, медичний персонал тощо) не можуть розглядатися як окремий сегмент ринку, оскільки їх зацікавленість є опосередкованою, а вплив на розвиток продукту невеликий.

Клієнти, яких зацікавить даний продукт, швидше за все, будуть відноситись до категорій або B2B, або B2G [47].

Натомість, запропоноване рішення може зацікавити державні та приватні медичні установи, дослідницькі організації, а також виробників відповідного апаратного забезпечення. Дані сегменти мають великий вплив та зацікавленість і їх слід вважати основними складовими цільового ринку.

Тому слід зосередити зусилля на приваблені вище згаданих сегментів та застосовувати відповідні розроблені стратегії у взаємовідносинах із ними (див. табл. 5.1).

5.6. Унікальна ціннісна пропозиція

Програмне забезпечення для візуалізації тривимірних медичних зображень повинне задовольняти цілому ряду вимог.

Однією із таких вимог є можливість перегляду зрізів тривимірних медичних зображень під довільними кутами зрізу, адже для вивчення внутрішнього стану об’єкту недостатньо простого перегляду зрізів, що лежать в ортогональних площинах.

Виконання даної вимоги є достатньо нетривіальною задачею, оскільки особливістю тривимірних медичних зображень є, як правило, їх великий розмір, та точність із якою зрізи даного зображення мають бути візуалізовані. При цьому, дану вимогу не можна відкинути через специфічність медичної галузі, де якісний аналіз при діагностиці є необхідною умовою.

Для вирішення даної задачі використовуються спеціальні підходи, зокрема, апаратні оптимізації, спеціальні алгоритми растеризації та методи візуалізації. Метою застосування вище вказаних методів є досягнення компромісу між програмною швидкодією та наочністю і точністю візуалізації медичних зображень.

В результаті запропоновано програмний продукт, який задовольняє вище означеним вимогам.

Представлене рішення надає кінцевому користувачеві простоту у використанні, високу якість візуалізації, високий рівень швидкодії, доступність та легкість у супроводі.

Продукт має відносно невисоку вартість у порівнянні із сучасними аналогічними рішеннями.

Запропоноване рішення принесе економічний ефект споживачам, оскільки його використання, крім відмови від додаткового апаратного забезпечення, дозволить зменшити кількість проведених досліджень та експериментів, спростить діагностику при лікуванні та встановлені діагнозів.

5.7. Доходи і витрати

Основним джерелом доходу є продаж платних версій продукту та надання ліцензій на використання, а також послуг із підтримки.

Пропонується розробити декілька версій програмного забезпечення, що будуть відрізнятися доступними функціональними можливостями:

- базова версія – надає користувачеві основні засоби для роботи із тривимірними зображеннями;
- розширена – користувач матиме змогу використовувати всі функціональні можливості продукту, проте на деякі із них будуть накладені певні обмеження;
- повна – користувачеві доступні усі функціональні можливості без обмежень на використання.

Кожна із запропонованих версій буде відноситися до окремої цінової категорії, для розширеної та повної версії будуть надаватися додаткові послуги із технічної підтримки.

Для розробки продукту та його супроводу передбачаються наступні витрати:

- витрати на розробку продукту, його супровід та підтримку (придбання необхідного апаратного та програмного забезпечення, закупівля необхідних ресурсів та оплата послуг);
- витрати на рекламу та маркетинг;
- витрати на оплату праці;
- адміністративні витрати (послуги банків, витрати на зв’язок);
- витрати на забезпечення інфраструктури (комунальні послуги, оренду приміщень);
- інші.

Прогнози доходів і витрат, а також розрахунок маржинальних прибутків [48] протягом 12 календарних місяців наведено у Додатку 1.

5.8. Бізнес-модель

В результаті проведеного аналізу були зібрані дані для побудови бізнес-моделі, які можна подати наступним чином, відповідно до канви бізнес-моделі [49]:

- проблема – складність дослідження тривимірних медичних зображень, обмеженість перегляду моделі у трьох площинах та ін.;
- рішення – програмне забезпечення для візуалізації тривимірних медичних зображень;
- унікальна ціннісна пропозиція – можливість візуалізації тривимірних медичних зображень великого обсягу із можливістю дослідження зрізів моделі під довільними кутами;
- споживачі – державні та приватні медичні заклади, науково-дослідні лабораторії, інститути та організації, навчальні заклади медичного спрямування;
- структура витрат;
- потоки доходів.

Подана канва містить розділи, такі як прихована перевага, ключові метрики, та канали збуту, відповідно до різновиду канви бізнес-моделі lean canvas [50].

Описані вище розділи представлені у Додатку 2.

З огляду на отримані результати, для вирішення окреслених проблем пропонується створити продукт, який представляє собою апаратно незалежне програмне забезпечення для візуалізації зрізів тривимірних медичних зображень під довільними кутами. Конкурентною перевагою даного продукту у порівнянні із аналогами буде використання розроблених алгоритмів для растеризації тривимірних зображень. Орієнтовними споживачами є медичні заклади та установи, а також дослідницькі

організації. Основним джерелом прибутку є продаж продукту, а також надання платних послуг по обслуговуванню та продовження терміну дії придбаних ліцензій.

5.9. Висновки

Запропонована бізнес-модель має забезпечити позитивний економічний ефект та вирішити виявлені проблеми дослідження та візуалізації тривимірних медичних зображень. Для цього пропонується розробити відповідне програмне забезпечення, що задовольнятиме вище зазначеним вимогам та вирішуватиме поставлені задачі.

Дане програмне забезпечення вносить на ринок унікальну ціннісну пропозицію та має свої конкурентні переваги у порівнянні із аналогами.

Розроблена модель потребує додаткового, більш глибокого, аналізу та доробки. Проте, виходячи із попередніх розрахунків, розробка та реалізація запропонованого програмного продукту є рентабельною і здатна покрити витрати на його створення та почати приносити дохід вже через півроку після завершення розробки.

ВИСНОВКИ

У ході проведення дослідження, представленого у даній роботі, була вирішена задача растеризації довільно орієнтованих зрізів тривимірних медичних зображень.

Було здійснено ґрунтовне вивчення та проведено аналіз особливостей роботи із тривимірними медичними зображеннями та воксельними моделями даних загалом. Після проведення дослідження різних методів та алгоритмів растеризації та візуалізації воксельних моделей даних, була поставлена задача, яка полягала у розробці власного методу растеризації зрізів.

Запропонований метод базується на принципі використання ткацьких технологій растеризації. Головною проблемою при растеризації зрізів воксельних моделей даних є дискретна природа таких даних, через що абсолютно точне представлення довільно орієнтованих зрізів тривимірного зображення є досить нетривіальною задачею. При цьому, для досягнення максимально точного результату растеризації, неминучим є падіння швидкодії обробки воксельних даних, особливо для тривимірних медичних зображень, які можуть характеризуватися великою розмірністю.

Завдяки мінімізації кількості обчислень в циклах растеризації фрагментів master-лінії, що досягається за рахунок визначення граничних координат для кожного фрагменту master-лінії, а також використанню лише цілочисельної арифметики, даний метод дозволяє виконувати растеризацію зрізів більш ефективно.

Порівнюючи швидкодію алгоритму розробленого методу із роботою точного ткацького алгоритму растеризації, було встановлено, що запропонований алгоритм працює швидше на 3-5%. При цьому, в половині випадків, розбіжності в точності растеризації відсутні. В інших ситуаціях результати растеризації майже не відрізняються візуально.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Estimating outdoor advertising media visibility with voxel-based approach [Text] / Sz. Chmielewski, P. Tompalski. // Applied Geography. — 2017. — Vol. 87. — P. 1–13.
2. Planning and rehearsal of surgical interventions in the volume model [Text] / B.Pflesser, R. Leuwer, U. Tiede, K. Hohne. // Medicine Meets Virtual Reality 2000. — 2000. — P. 259–264.
3. Kaufman, A. Volume visualization and volume graphics [Text] / A. Kaufman, K. Mueller. // Stony Brook University, Stony Brook. – 2003. — P. 10–15.
4. Dachsbacher, C. Rendering Procedural Terrain by Geometry Image Warping [Text] / C. Dachsbacher, M. Stamminger. // Eurographics Symposium on Rendering. — 2004. — P. 103–110.
5. Kaufman, A. Volume visualization: Principles and advances [Text] / Arie Kaufman. // Course notes. — 1997. — Vol. 24.
6. Laine, S. Efficient sparse voxel octrees [Text] / S. Laine, T. Karras. // IEEE Transactions on Visualization and Computer Graphics. — 2011. — Vol. 17, №8. — P. 1048–1059.
7. Mazura, A. Virtual Cutting in Medical Data [Text] / A. Mazura, S. Seifert. // Medicine Meets Virtual Reality. — San Diego, CA, 1997.
8. Udupa, J. 3D imaging in medicine [Text] / J. Udupa, G. Herman. — 2nd edition. — CRC Press, 1999. — 384 p.
9. Бурых, М. П. Воксельное анатомическое моделирование внутренних органов человека [Текст] / М. П. Бурых, Р.С. Ворощук // Клінічна анатомія та оперативна хірургія. — 2006. — Т. 5, № 4. — С.115–118.
10. Webb, R. Fundamentals of Body CT [Text] / R. Webb, W. Brant, N. Major. — 4th edition. — Elsevier Health Sciences, 2014. — 434 p.
11. Characteristics and Control of Contrast in CT [Електронний ресурс] — Режим доступу : https://pubs.rsna.org/doi/pdf/10.1148

/radiographics.12.4.1636042 — Дата доступу : Травень 2018. — Назва з екрана.
12. Cobbold, R. Foundations of Biomedical Ultrasound [Text] / Richard S. C. Cobbold. — Oxford University Press, 2006. — 832 p.
13. Хорнак Дж. П. Основы МРТ [Електронний ресурс] — Режим доступу : https://www.cis.rit.edu/htbooks/mri/ — Дата доступу : Травень 2018. — Назва з екрана.
14. Digital Imaging and Communications in Medicine [Електронний ресурс] — Режим доступу : https://www.dicomstandard.org/current/ — Дата доступу : Травень 2018. — Назва з екрана.
15. The DICOM Standard [Електронний ресурс] — Режим доступу : http://www.cabiatl.com/mricro/dicom/index.html — Дата доступу : Травень 2018. — Назва з екрана.
16. The NIFTI file format [Електронний ресурс] — Режим доступу : https://brainder.org/2012/09/23/the-nifti-file-format/ — Дата доступу : Травень 2018. — Назва з екрана.
17. The NIFTI-2 file format [Електронний ресурс] — Режим доступу : https://brainder.org/2015/04/03/the-nifti-2-file-format/ — Дата доступу : Травень 2018. — Назва з екрана.
18. Zhang, Q. Volume visualization: a technical overview with a focus on medical applications [Text] / Q. Zhang, R. Eagleson, T. Peters. // Journal of digital imaging. — 2011. — Vol. 24, №4. — P. 640–664.
19. Меркулова, Е. В. Создание алгоритмов построения трехмерной воксельной модели на основании результатов СКТ [Текст] / Меркулова Е. В., Адамов В. Г., Кондратов В. И. // Сборник научных трудов Sworld / гл. ред. Куприенко С. В. — Иваново, 2015. — Вып. 1, т. 2. — С. 72–79.
20. Volume Visualization – Computer Graphics Laboratory ETH Zurich [Електронний ресурс] — Режим доступу : https://graphics.ethz.ch/teaching /former/scivis_07/Notes/stuff/StuttgartCourse/VIS-Modules-05-

Volume_Visualization.pdf — Дата доступу : Травень 2018. — Назва з екрана.

21. Lorensen, W. Marching cubes: A high resolution 3D surface construction algorithm [Text] / W. Lorensen, H. Cline. // ACM siggraph computer graphics. — 1987. — Vol. 21, №4. — P. 163–169.
22. Westover, L. SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm [Text] / Westover Lee Alan — Chapel Hill, 1991. — 103 p.
23. Lacroute, P. Fast volume rendering using a shear-warp factorization of the viewing transformation [Text] / P. Lacroute, M. Levoy. // Proceeding SIGGRAPH '94 Proceedings of the 21st annual conference on Computer graphics and interactive technique. — 1994. — P. 451–458.
24. An Extended Volume Visualization System for Arbitrary Parallel Projection [Text] / R.Bakalash, A. Kaufman, R. Pacheco, H. Pfister. // Eurographics Workshop on Graphics Hardware. — 1992. — P. 64–69.
25. Chen, J. Multiple Segment Line Scan-conversion [Text] / Jim Chen. // Computer Graphics Forum. — 1997. — №5. — P. 257–268.
26. Kaufman, A. 3D scan-conversion algorithms for voxel-based graphics [Text] / A. Kaufman, E. Shimony. // Proceedings of the 1986 workshop on Interactive 3D graphics. — 1987. — P. 45–75.
27. Kaufman, A. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes [Text] / Arie Kaufman. // ACM Siggraph Computer Graphics. — 1987. — Vol. 21, №4. — P. 171–179.
28. Computer graphics: Principles and Practice [Text] / Hughes, et al. — 3rd edition. — Addison-Wesley, 2013. — 1209 p.
29. Fast Arbitrarily Oriented 3D Discrete Slicing for Medical Imaging [Електронний ресурс] — Режим доступу : https://www.semanticscholar.org/paper/Fast-Arbitrarily-Oriented-3D-Discrete-Slicing-for-Guitton — Дата доступу : Травень 2018. — Назва з екрана.

30. Kaufman, A. An algorithm for 3D scan-conversion of polygons [Text] / Arie Kaufman. // Eurographics Association. — 1987.
31. Klette, R. The m-dimensional grid point space [Text] / Reinhard Klette. // Computer Vision, Graphics, and Image Processing. — 1985. — Vol. 30, №1. — P. 1–12.
32. Wüthrich, C. A model for curve rasterization in n-dimensional space [Text] / Charles Wüthrich. // Computers & Graphics. — 1998. — Vol. 22. — P. 153–160.
33. Lincke, C. An Exact Weaving Rasterization Algorithm For Digital Planes [Text] / C. Lincke, C. Wüthrich, P. Guitton. // WSCG'99. — 1999.
34. Freeman, H. On the Encoding of Arbitrary Geometric Configurations [Text] / Herbert Freeman. // IRE Transactions on Electronic Computers. — 1961. — Vol. 10, №2. — P. 260–268.
35. Pham, S. Equations of Digital Straight Lines [Text] / Son Pham. // In: Kunii T.L. (eds) Computer Graphics 1987. Springer, Tokyo. — 1987. — P. 221–248.
36. Petković, T. Supercover Plane Rasterization: A Rasterization Algorithm for Generating Supercover Plane Inside a Cube [Text] / T. Petković, S. Lončarić. // 2nd International Conference on Computer Graphics Theory and Applications. — 2007.
37. Cohen, D. Fundamentals of surface voxelization [Text] / D. Cohen, A. Kaufman. // Graphical models and image processing. — 1995. — Vol. 56, №6. — P. 453–461.
38. Cohen, D. Scan-conversion algorithms for linear and quadratic objects [Text] / D. Cohen, A. Kaufman. // Volume visualization. — 1990. — Vol. 280. — P. 301.
39. Bresenham’s Algorithm [Електронний ресурс] — Режим доступу : http://www.idav.ucdavis.edu/education/GraphicsNotes/Bresenhams-Algorithm.pdf — Дата доступу : Травень 2018. — Назва з екрана.

40. Fujimoto, A. ARTS: Accelerated Ray-Tracing System [Text] / A. Fujimoto, T. Tanaka, K. Iwata. // IEEE Computer Graphics and Applications. — 1986. — Vol. 6, №4. — P. 16–26.
41. Страуструп, Б. Язык программирования C++ [Текст] / Б. Страуструп : пер. с англ. — СПб.; М. : Невский диалект — Бином, 1999. — 991 с.
42. Шилдт, Г. C#: полное руководство [Текст] / Г. Шилдт : пер. с англ. — М. : ООО «И.Д. Вильямс», 2011. — 1056 с.
43. Deitel, P. Java How to Program, Early Objects, 11th Edition [Text] / P. Deitel, H. Deitel. — 11th edition. — Pearson, 2017. — 1296 p.
44. Алгоритм DDA-линии [Електронний ресурс] — Режим доступу : http://compgraphics.info/2D/DDA_line.php — Дата доступу : Травень 2018. — Назва з екрана.
45. Bresenham, J. Algorithm for computer control of a digital plotter [Text] / Jack Bresenham. // IBM Systems Journal. — 1965. — Vol. 4, №1. — P. 25–30.
46. Interactivity is the key [Електронний ресурс] — Режим доступу : http://www.ssec.wisc.edu/~billh/p39-hibbard.pdf — Дата доступу : Травень 2018. — Назва з екрана.
47. Бек, М. А. Маркетинг B2B [Текст] / М. А. Бек ; Гос. ун-т — Высшая школа экономики. — М. : Изд. дом ГУ ВШЭ, 2008. — 327 с.
48. Базылев, Н. Экономическая теория [Текст] / Н. Базылев, М. Базылева — М.: Результат и Качество, 2010. — С. 640.
49. Остервальдер, А. Построение бизнес-моделей: Настольная книга стратега и новатора [Текст] / А. Остервальдер, И. Пинье ; — Альпина Паблишер. — 2017. — 288 с.
50. Бизнес с нуля: Метод Lean Startup для быстрого тестирования идей и выбора бизнес-модели [Текст] / Эрик Рис. — М.: Альпина Паблишер, 2012. — 253 с.

ДОДАТКИ

Додаток 1

Прогнози доходів і витрат. Маржинальні прибутки

Прогноз доходів

№	Категорія	Доходи за місяць (у. о.)											
		1	2	3	4	5	6	7	8	9	10	11	12
1	Продаж базової версії	-	-	-	-	-	-	-	-	-	-	-	-
2	Продаж розширеної версії	-	-	-	-	2000	2000	2000	2000	2000	4000	4000	4000
3	Продаж повної версії	-	-	-	-	1000	1000	1000	2000	2000	4000	4000	8000
4	Надання платних послуг	-	-	-	-	1000	1000	2000	2000	2000	2000	2000	2000
5	Продовження ліцензій	-	-	-	-	-	-	2000	2000	2000	4000	8000	8000
	Всього	-	-	-	-	4000	4000	7000	8000	8000	14000	18000	22000

Прогноз витрат

№	Категорія	Витрати за місяць (у. о.)											
		1	2	3	4	5	6	7	8	9	10	11	12
1	Розробка продукту	7000	7000	5000	2500	-	-	-	-	-	-	-	-
2	Супровід та підтримка	-	-	-	-	500	500	500	500	500	500	500	500
3	Реклама та маркетинг	2000	2000	1000	1000	500	500	500	500	500	500	500	500
4	Оплата праці	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000	10000
5	Адміністративні витрати	500	500	500	500	500	500	500	500	500	500	500	500
6	Забезпечення інфраструктури	500	500	500	500	500	500	500	500	500	500	500	500
7	Інші	200	200	200	200	200	200	200	200	200	200	200	200
	Всього	20200	20200	17200	14700	12200	12200	12200	12200	12200	12200	12200	12200

Маржинальні прибутки

Місяць	1	2	3	4	5	6	7	8	9	10	11	12
Прибуток (у. о.)	-20200	-20200	-17200	-14700	-8200	-8200	-5200	-4200	-4200	1800	5800	9800

Додаток 2

Канва бізнес-моделі

Канва бізнес-моделі

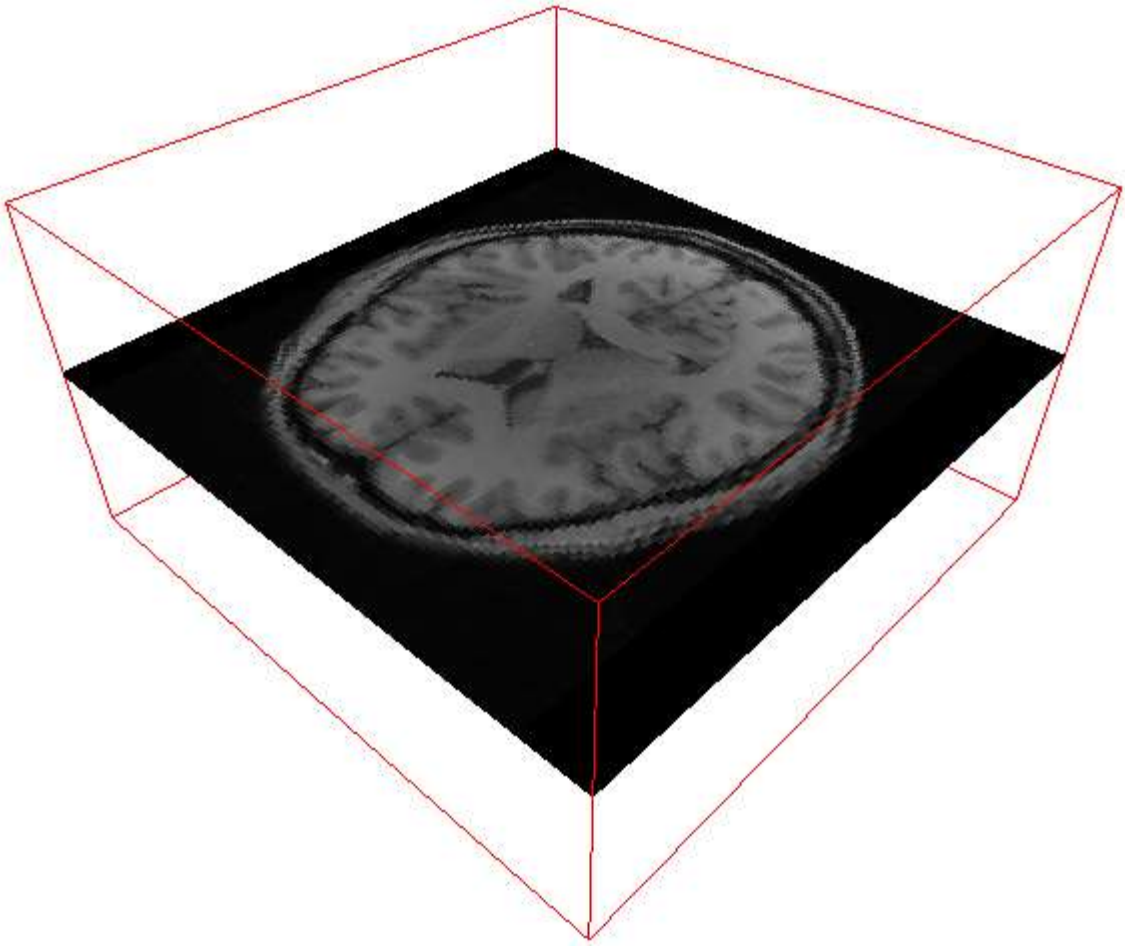
<table>
<tr>
<td>Проблема
– складність дослідження тривимірних медичних зображень;
– обмеженість перегляду моделі у трьох площинах.</td>
<td>Рішення
Програмне забезпечення для візуалізації тривимірних зображень.

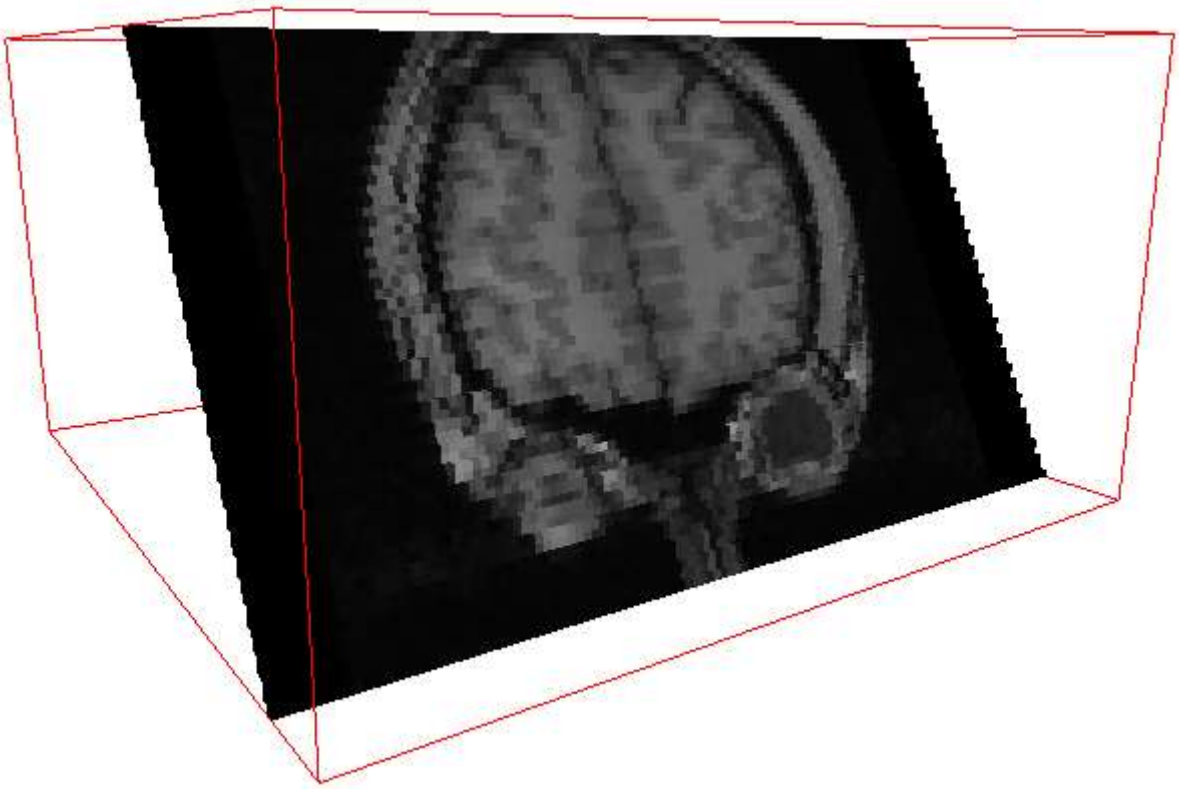
Ключові метрики
– обсяг продажу продукту;
– кількість підключень платних функціональних можливостей;
– кількість звернень до техпідтримки;
– кількість подовжених ліцензій.</td>
<td>Унікальна ціннісна пропозиція
Можливість візуалізації тривимірних медичних зображень великого обсягу із можливістю дослідження зрізів моделі під довільними кутами.</td>
<td>Прихована перевага
Використання розроблених методів для растеризації тривимірних зображень.

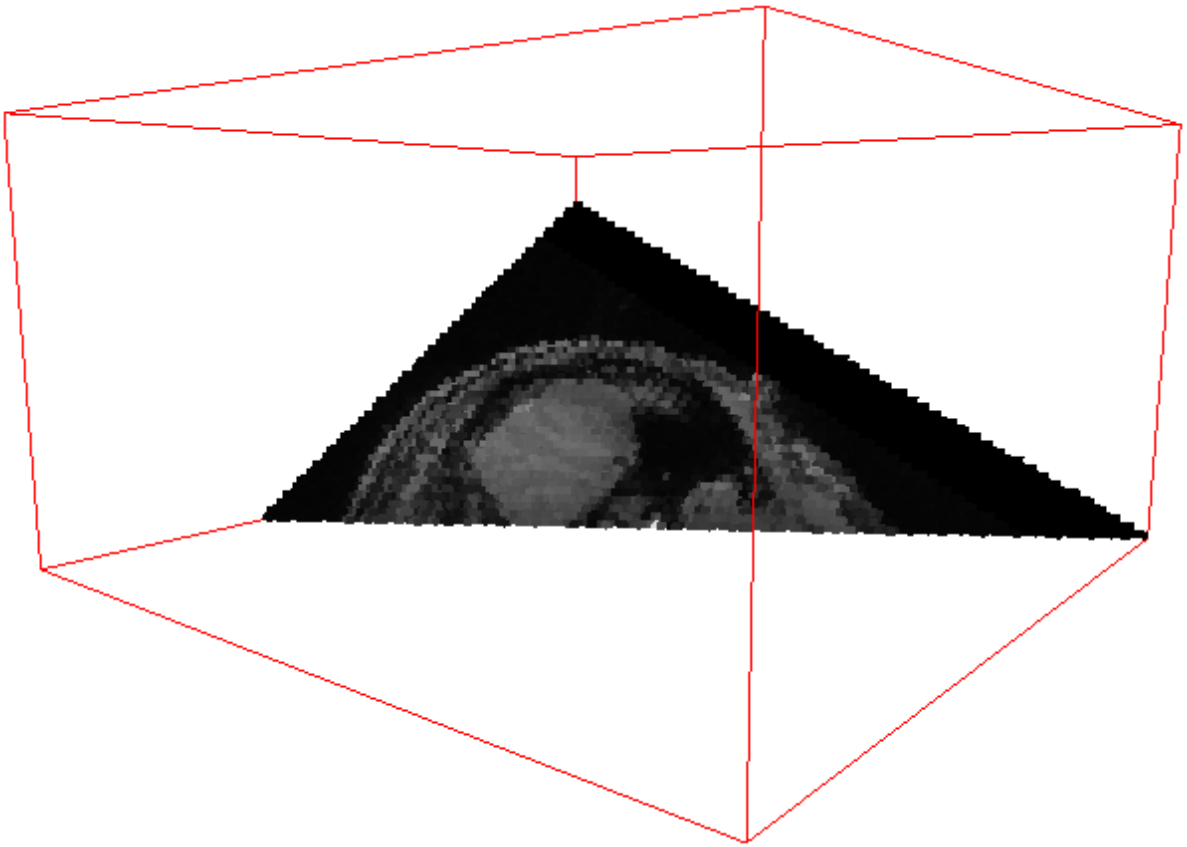
Канали збуту
– власний web-ресурс;
– безпосередні контакти із замовниками.</td>
<td>Споживачі
– державні та приватні медичні заклади;
– науково-дослідні лабораторії, інститути та організації;
– навчальні заклади медичного спрямування.</td>
</tr>
<tr>
<td colspan="3">Структура витрат
– розробка продукту;
– супровід та підтримка;
– оплата праці;
– реклама та маркетинг;
– адміністративні витрати;
– підтримка інфраструктури;
– інші.</td>
<td colspan="2">Потоки доходів
– продаж розширеної та повної версій продукту;
– надання платних послуг;
– продовження ліценцій.</td>
</tr>
</table>

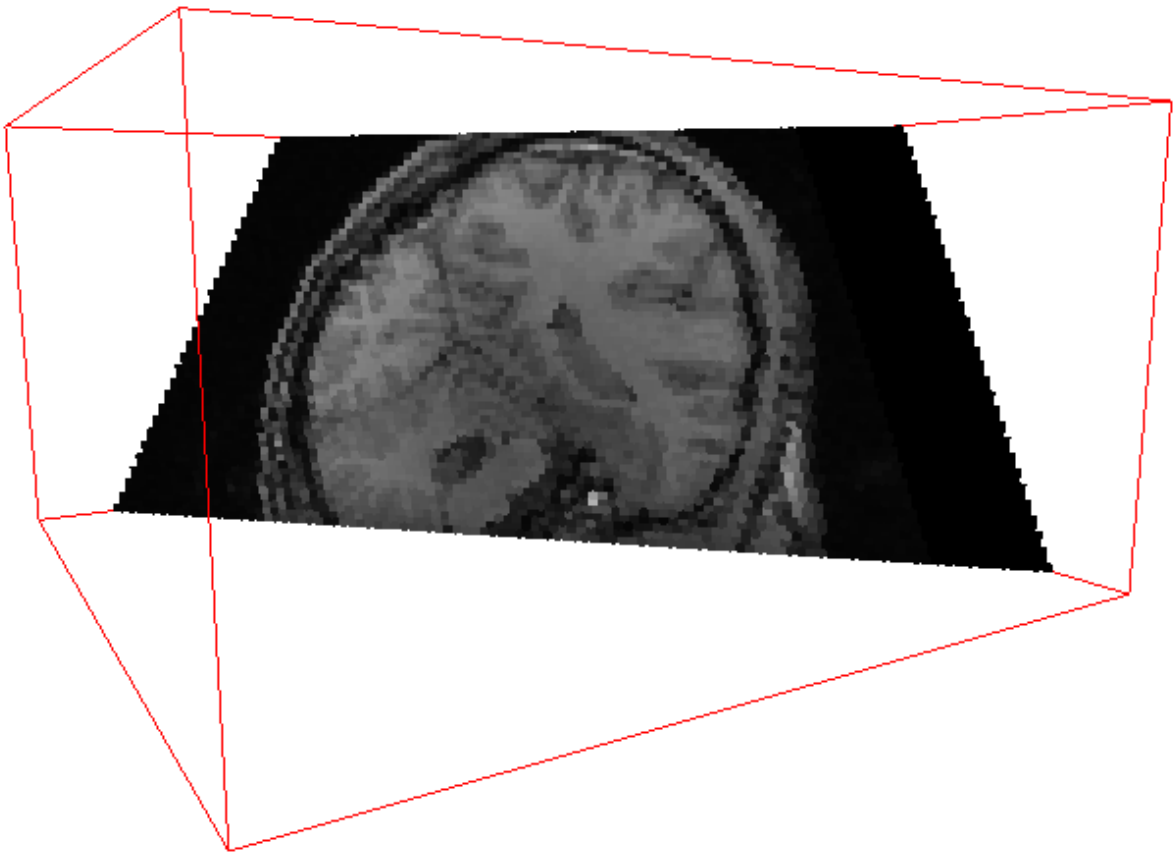
Додаток З

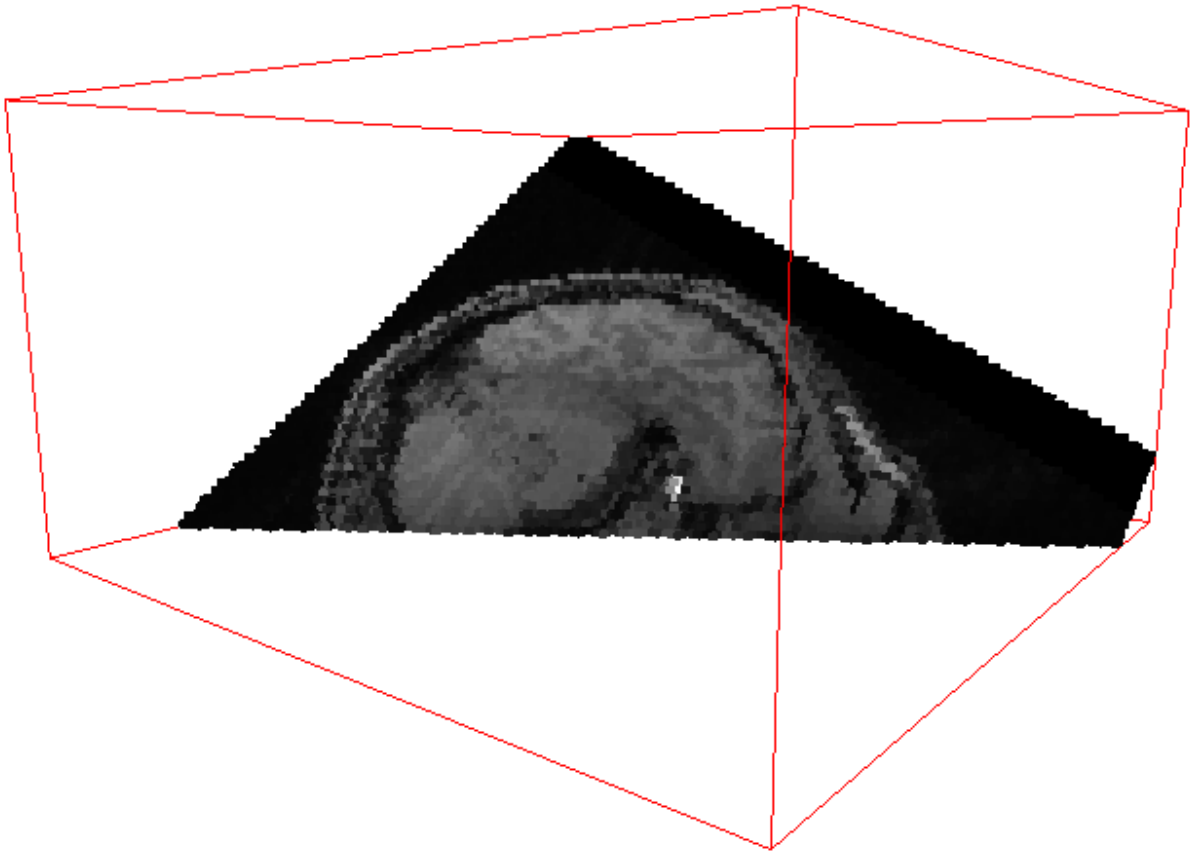
Результати растеризації тривимірних медичних зображень

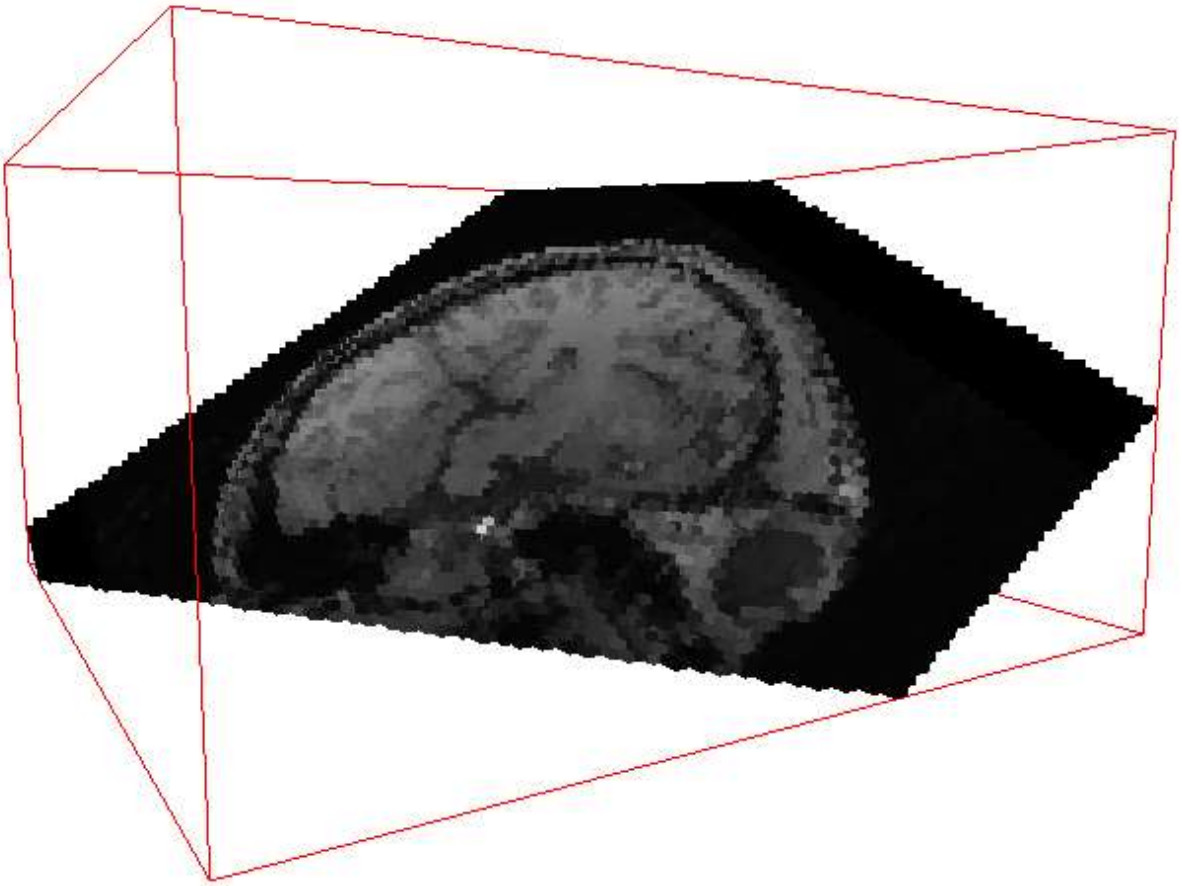












Додаток 4

Лістинги

Лістинг 1. Методи класу Slicer для растеризації зрізів тривимірних зображень

```
point_vec SlicerLSWR::getSlice(const Plane<double>& plane) const {
    point_set vertices = getSliceVertices(plane);

    switch (vertices.size()) {
        case 0: return point_vec();
        case 1: return rasterizeVertex(vertices);
        case 2: return rasterizeEdge(vertices);
        case 3: return rasterizeTriangle(vertices);
        case 4: return rasterizeTetragon(plane, vertices);
        case 5: return rasterizePentagon(plane);
        case 6: return rasterizeHexagon(plane);
        default:
            throw std::logic_error("Slice geometry has more than 6
vertices.");
    }
}

point_set SlicerLSWR::getSliceVertices(const Plane<double>& plane) const {
    point_set vertices;
    Point<int> modelEdge = model->getEdge();

    if (plane.getA() != 0.0) {
        Point<int> p1 = plane.solve(0.0, 0.0, 'x');
        Point<int> p2 = plane.solve(0.0, modelEdge.z, 'x');
        Point<int> p3 = plane.solve(modelEdge.y, 0.0, 'x');
        Point<int> p4 = plane.solve(modelEdge.y, modelEdge.z, 'x');
        if (model->contains(p1)) vertices.insert(p1);
        if (model->contains(p2)) vertices.insert(p2);
        if (model->contains(p3)) vertices.insert(p3);
        if (model->contains(p4)) vertices.insert(p4);
    }

    if (plane.getB() != 0.0) {
        Point<int> p1 = plane.solve(0.0, 0.0, 'y');
        Point<int> p2 = plane.solve(0.0, modelEdge.z, 'y');
        Point<int> p3 = plane.solve(modelEdge.x, 0.0, 'y');
        Point<int> p4 = plane.solve(modelEdge.x, modelEdge.z, 'y');
        if (model->contains(p1)) vertices.insert(p1);
        if (model->contains(p2)) vertices.insert(p2);
        if (model->contains(p3)) vertices.insert(p3);
        if (model->contains(p4)) vertices.insert(p4);
    }

    if (plane.getC() != 0.0) {
        Point<int> p1 = plane.solve(0.0, 0.0, 'z');
        Point<int> p2 = plane.solve(0.0, modelEdge.y, 'z');
        Point<int> p3 = plane.solve(modelEdge.x, 0.0, 'z');
        Point<int> p4 = plane.solve(modelEdge.x, modelEdge.y, 'z');
        if (model->contains(p1)) vertices.insert(p1);
        if (model->contains(p2)) vertices.insert(p2);
        if (model->contains(p3)) vertices.insert(p3);
        if (model->contains(p4)) vertices.insert(p4);
    }

    return vertices;
}
```

```
point_vec SlicerLSWR::sortSliceVertices(const point_set& vertices) const {
    if (vertices.size() < 4) {
        return point_vec(vertices.begin(), vertices.end());
    }

    Point<int> modelEdge = model->getEdge();
    point_lst unsorted(++vertices.begin(), vertices.end());
    point_vec sorted(vertices.begin(), ++vertices.begin());

    while (unsorted.size()) {
        Point<int> lastSortedPoint = sorted.back();

        for (point_lst::iterator i = unsorted.begin();
             i != unsorted.end();
             ++i) {
            Point<int> unsortedPoint = *i;

            if (unsortedPoint.x == lastSortedPoint.x) {
                if (unsortedPoint.x == 0
                    || unsortedPoint.x == modelEdge.x) {
                    sorted.push_back(unsortedPoint);
                    unsorted.erase(i);
                    break;
                }
            }
            if (unsortedPoint.y == lastSortedPoint.y) {
                if (unsortedPoint.y == 0
                    || unsortedPoint.y == modelEdge.y) {
                    sorted.push_back(unsortedPoint);
                    unsorted.erase(i);
                    break;
                }
            }
            if (unsortedPoint.z == lastSortedPoint.z) {
                if (unsortedPoint.z == 0
                    || unsortedPoint.z == modelEdge.z) {
                    sorted.push_back(unsortedPoint);
                    unsorted.erase(i);
                    break;
                }
            }
        }
    }

    return sorted;
}

point_vec SlicerLSWR::rasterizeTetragon(const Plane<double>& plane,
                                        const point_set& vertices) const {
    point_vec sorted = sortSliceVertices(vertices);
    Point<int> v1 = sorted[0];
    Point<int> v2 = sorted[1];
    Point<int> v3 = sorted[2];

    if (v1.x == v2.x && v2.x == v3.x) {
        return rasterizeOrthogonalX(v1.x);
    }
    if (v1.y == v2.y && v2.y == v3.y) {
        return rasterizeOrthogonalY(v1.y);
    }
    if (v1.z == v2.z && v2.z == v3.z) {
        return rasterizeOrthogonalZ(v1.z);
    }
```

```
    if ((v1.x == v2.x && v1.y == v2.y) || (v3.x == v2.x && v3.y == v2.y)) {
        return rasterizeParallelZ(v1, v2, v3);
    }
    if ((v1.y == v2.y && v1.z == v2.z) || (v3.y == v2.y && v3.z == v2.z)) {
        return rasterizeParallelX(v1, v2, v3);
    }
    if ((v1.x == v2.x && v1.z == v2.z) || (v3.x == v2.x && v3.z == v2.z)) {
        return rasterizeParallelY(v1, v2, v3);
    }

    return rasterizeTetragon(plane);
}

point_vec SlicerLSWR::rasterizeTetragon(const Plane<double>& plane) const {
    return rasterize(plane);
}

point_vec SlicerLSWR::rasterizePentagon(const Plane<double>& plane) const {
    return rasterize(plane);
}

point_vec SlicerLSWR::rasterizeHexagon(const Plane<double>& plane) const {
    return rasterize(plane);
}

point_vec SlicerLSWR::rasterizeOrthogonalX(int offset) const {
    int length = model->getLength();
    int height = model->getHeight();
    int width = model->getWidth();

    if (offset < 0 || offset > length) {
        throw std::logic_error("X offset is out of model.");
    }

    point_vec points;

    for (int y = 0; y < height; ++y) {
        for (int z = 0; z < width; ++z) {
            points.push_back(Point<int>(offset, y, z));
        }
    }

    return points;
}

/*
 * @note: Second vertex is considered as the beginning of the master
 * and base lines.
 */
point_vec SlicerLSWR::rasterizeParallelX(const Point<int>& v1,
                                         const Point<int>& v2,
                                         const Point<int>& v3) const {
    int length = model->getLength();
    point_vec points;

    if (v3.y == v2.y && v3.z == v2.z) {
      drawing::BresenhamsLineAlgorithm(v1.y,v1.z,v2.y,v2.z,[&](int y,int z)
        {
            for (int x = 0; x < length; ++x) {
                points.push_back(Point<int>(x, y, z));
            }
        });
    }
```

```
    else if (v1.y == v2.y && v1.z == v2.z) {
      drawing::BresenhamsLineAlgorithm(v2.y,v2.z,v3.y,v3.z,[&](int y,int z)
        {
            for (int x = 0; x < length; ++x) {
                points.push_back(Point<int>(x, y, z));
            }
        });
    } else {
        throw std::logic_error("Slice is not parallel to X axis.");
    }

    return points;
}

point_vec SlicerLSWR::rasterize(const Plane<double>& plane) const {
    Point<int> start, masterEnd, baseEnd;
    int direction =
        defineMasterAndBaseLines(plane, start, masterEnd, baseEnd);

    return rasterize(start, masterEnd, baseEnd, direction);
}

point_vec SlicerLSWR::rasterize(const Point<int>& start,
                                const Point<int>& masterEnd,
                                const Point<int>& baseEnd,
                                int direction) const {
    if (direction < 0) {
        direction = defineLinesDirection(start, masterEnd, baseEnd);
    }

    switch (direction) {
        case XY_XZ: return rasterize_XY_XZ(start, masterEnd, baseEnd);
        case YX_YZ: return rasterize_YX_YZ(start, masterEnd, baseEnd);
        case XZ_XY: return rasterize_XZ_XY(start, masterEnd, baseEnd);
        case ZX_ZY: return rasterize_ZX_ZY(start, masterEnd, baseEnd);
        case YZ_YX: return rasterize_YZ_YX(start, masterEnd, baseEnd);
        case ZY_ZX: return rasterize_ZY_ZX(start, masterEnd, baseEnd);
        default:
            throw std::logic_error("Unknown rasterization direction.");
    }
}

int SlicerLSWR::defineMasterAndBaseLines(const Plane<double>& plane,
                                         Point<int>& start,
                                         Point<int>& masterEnd,
                                         Point<int>& baseEnd) const {
    Point<int> modelEdge = model->getEdge();
    Point<int> _start, _masterEnd, _baseEnd;
    Point<int> v1, v2, v3;
    int orientation, _orientation;
    double masterLineLength, _masterLineLength;

    defineBaseTriangle(plane, v1, v2, v3);

    // v1v2 considered as the master line.
    if (abs(v1.x - v2.x) < abs(v1.y - v2.y)) {
        orientation = XY_XZ;
        start = v1;
        masterEnd = (v2.y > modelEdge.y) ?
            (Point<int>) plane.solve(modelEdge.y, v2.z, 'x') : (v2.y < 0) ?
            (Point<int>) plane.solve(0.0, v2.z, 'x') :
            v2;
```

```
    baseEnd = (v3.z > modelEdge.z) ?
        (Point<int>) plane.solve(v3.y, modelEdge.z, 'x') : (v3.z < 0) ?
        (Point<int>) plane.solve(v3.y, 0.0, 'x') :
        v3;
} else {
    orientation = YX_YZ;
    start = v2;
    masterEnd = (v1.x > modelEdge.x) ?
        (Point<int>) plane.solve(modelEdge.x, v1.z, 'y') : (v1.x < 0) ?
        (Point<int>) plane.solve(0.0, v1.z, 'y') :
        v1;
    baseEnd = (v3.z > modelEdge.z) ?
        (Point<int>) plane.solve(v3.x, modelEdge.z, 'y') : (v3.z < 0) ?
        (Point<int>) plane.solve(v3.x, 0.0, 'y') :
        v3;
}

masterLineLength = math::calculateDistance(start, masterEnd);

// v1v3 considered as the master line.
if (abs(v1.x - v3.x) < abs(v1.z - v3.z)) {
    _orientation = XZ_XY;
    _start = v1;
    _masterEnd = (v3.z > modelEdge.z) ?
        (Point<int>) plane.solve(v3.y, modelEdge.z, 'x') : (v3.z < 0) ?
        (Point<int>) plane.solve(v3.y, 0.0, 'x') :
        v3;
    _baseEnd = (v2.y > modelEdge.y) ?
        (Point<int>) plane.solve(modelEdge.y, v2.z, 'x') : (v2.y < 0) ?
        (Point<int>) plane.solve(0.0, v2.z, 'x') :
        v2;
} else {
    _orientation = ZX_ZY;
    _start = v3;
    _masterEnd = (v1.x > modelEdge.x) ?
        (Point<int>) plane.solve(modelEdge.x, v1.y, 'z') : (v1.x < 0) ?
        (Point<int>) plane.solve(0.0, v1.y, 'z') :
        v1;
    _baseEnd = (v2.y > modelEdge.y) ?
        (Point<int>) plane.solve(v2.x, modelEdge.y, 'z') : (v2.y < 0) ?
        (Point<int>) plane.solve(v2.x, 0.0, 'z') :
        v2;
}

_masterLineLength = math::calculateDistance(_start, _masterEnd);

if (_masterLineLength < masterLineLength) {
    masterLineLength = _masterLineLength;
    orientation = _orientation;
    masterEnd = _masterEnd;
    baseEnd = _baseEnd;
    start = _start;
}

// v2v3 considered as the master line.
if (abs(v2.y - v3.y) < abs(v2.z - v3.z)) {
    _orientation = YZ_YX;
    _start = v2;
    _masterEnd = (v3.z > modelEdge.z) ?
        (Point<int>) plane.solve(v3.x, modelEdge.z, 'y') : (v3.z < 0) ?
        (Point<int>) plane.solve(v3.x, 0.0, 'y') :
        v3;
```

```
            _baseEnd = (v1.x > modelEdge.x) ?
                (Point<int>) plane.solve(modelEdge.x, v1.z, 'y') : (v1.x < 0) ?
                (Point<int>) plane.solve(0.0, v1.z, 'y') :
                v1;
        } else {
            _orientation = ZY_ZX;
            _start = v3;
            _masterEnd = (v2.y > modelEdge.y) ?
                (Point<int>) plane.solve(v2.x, modelEdge.y, 'z') : (v2.y < 0) ?
                (Point<int>) plane.solve(v2.x, 0.0, 'z') :
                v2;
            _baseEnd = (v1.x > modelEdge.x) ?
                (Point<int>) plane.solve(modelEdge.x, v1.y, 'z') : (v1.x < 0) ?
                (Point<int>) plane.solve(0.0, v1.y, 'z') :
                v1;
        }

        _masterLineLength = math::calculateDistance(_start, _masterEnd);

        if (_masterLineLength < masterLineLength) {
            orientation = _orientation;
            masterEnd = _masterEnd;
            baseEnd = _baseEnd;
            start = _start;
        }

        return orientation;
    }

    int SlicerLSWR::defineLinesDirection(const Point<int>& start,
                                         const Point<int>& masterEnd,
                                         const Point<int>& baseEnd) const {
        int direction;

        if (start.x == masterEnd.x) {
            int elongationY = abs(start.y - masterEnd.y);
            int elongationZ = abs(start.z - masterEnd.z);
            direction = (elongationY < elongationZ) ? YZ_YX : ZY_ZX;

        } else if (start.y == masterEnd.y) {
            int elongationX = abs(start.x - masterEnd.x);
            int elongationZ = abs(start.z - masterEnd.z);
            direction = (elongationX < elongationZ) ? XZ_XY : ZX_ZY;

        } else if (start.z == masterEnd.z) {
            int elongationX = abs(start.x - masterEnd.x);
            int elongationY = abs(start.y - masterEnd.y);
            direction = (elongationX < elongationY) ? XY_XZ : YX_YZ;
        } else {
            throw std::logic_error("Invalid master-line ends.");
        }

        return direction;
    }
```

```
/*
 * @note: First vertex will be on the edge parallel to X axis;
 *        Second vertex will be on the edge parallel to Y axis;
 *        Third vertex will be on the edge parallel to Z axis.
 */
void SlicerLSWR::defineBaseTriangle(const Plane<double>& plane,
                                    Point<int>& v1,
                                    Point<int>& v2,
                                    Point<int>& v3) const {
    double a = plane.getA(), b = plane.getB(), c = plane.getC();
    double d = plane.getD();

    Point<int> _v1, _v2, _v3;

    if ((a >= 0.0 && b >= 0.0 && c >= 0.0 && d < 0.0) ||
        (a < 0.0 && b < 0.0 && c < 0.0 && d >= 0.0)) {
        v1 = plane.solve(0.0, 0.0, 'x');
        v2 = plane.solve(0.0, 0.0, 'y');
        v3 = plane.solve(0.0, 0.0, 'z');
        _v1 = plane.solve(model->getEdge().y, model->getEdge().z, 'x');
        _v2 = plane.solve(model->getEdge().x, model->getEdge().z, 'y');
        _v3 = plane.solve(model->getEdge().x, model->getEdge().y, 'z');

        if (math::calculateTriangleArea(v1, v2, v3)
            > math::calculateTriangleArea(_v1, _v2, _v3)) {
            v1 = _v1;
            v2 = _v2;
            v3 = _v3;
        }
    } else
    if ((a < 0.0 && b >= 0.0 && c >= 0.0 && d < 0.0) ||
        (a >= 0.0 && b < 0.0 && c < 0.0 && d >= 0.0)) {
        v1 = plane.solve(0.0, 0.0, 'x');
        v2 = plane.solve(model->getEdge().x, 0.0, 'y');
        v3 = plane.solve(model->getEdge().x, 0.0, 'z');
        _v1 = plane.solve(model->getEdge().y, model->getEdge().z, 'x');
        _v2 = plane.solve(0.0, model->getEdge().z, 'y');
        _v3 = plane.solve(0.0, model->getEdge().y, 'z');

        if (math::calculateTriangleArea(v1, v2, v3)
            > math::calculateTriangleArea(_v1, _v2, _v3)) {
            v1 = _v1;
            v2 = _v2;
            v3 = _v3;
        }
    } else
    if ((a >= 0.0 && b < 0.0 && c >= 0.0 && d < 0.0) ||
        (a < 0.0 && b >= 0.0 && c < 0.0 && d >= 0.0)) {
        v1 = plane.solve(model->getEdge().y, 0.0, 'x');
        v2 = plane.solve(0.0, 0.0, 'y');
        v3 = plane.solve(0.0, model->getEdge().y, 'z');
        _v1 = plane.solve(0.0, model->getEdge().z, 'x');
        _v2 = plane.solve(model->getEdge().x, model->getEdge().z, 'y');
        _v3 = plane.solve(model->getEdge().x, 0.0, 'z');

        if (math::calculateTriangleArea(v1, v2, v3)
            > math::calculateTriangleArea(_v1, _v2, _v3)) {
            v1 = _v1;
            v2 = _v2;
            v3 = _v3;
        }
    } else
```

```
if ((a >= 0.0 && b >= 0.0 && c < 0.0 && d < 0.0) ||
    (a < 0.0 && b < 0.0 && c >= 0.0 && d >= 0.0)) {
    v1 = plane.solve(0.0, model->getEdge().z, 'x');
    v2 = plane.solve(0.0, model->getEdge().z, 'y');
    v3 = plane.solve(0.0, 0.0, 'z');
    _v1 = plane.solve(model->getEdge().y, 0.0, 'x');
    _v2 = plane.solve(model->getEdge().x, 0.0, 'y');
    _v3 = plane.solve(model->getEdge().x, model->getEdge().y, 'z');

    if (math::calculateTriangleArea(v1, v2, v3)
        > math::calculateTriangleArea(_v1, _v2, _v3)) {
        v1 = _v1;
        v2 = _v2;
        v3 = _v3;
    }
} else
if ((a < 0.0 && b < 0.0 && c >= 0.0 && d < 0.0) ||
    (a >= 0.0 && b >= 0.0 && c < 0.0 && d >= 0.0)) {
    v1 = plane.solve(0.0, model->getEdge().y, 'x');
    v2 = plane.solve(model->getEdge().x, 0.0, 'y');
    v3 = plane.solve(model->getEdge().x, model->getEdge().y, 'z');
    _v1 = plane.solve(model->getEdge().y, 0.0, 'x');
    _v2 = plane.solve(0.0, model->getEdge().z, 'y');
    _v3 = plane.solve(0.0, 0.0, 'z');

    if (math::calculateTriangleArea(v1, v2, v3)
        > math::calculateTriangleArea(_v1, _v2, _v3)) {
        v1 = _v1;
        v2 = _v2;
        v3 = _v3;
    }
} else
if ((a < 0.0 && b >= 0.0 && c < 0.0 && d < 0.0) ||
    (a >= 0.0 && b < 0.0 && c >= 0.0 && d >= 0.0)) {
    v1 = plane.solve(0.0, model->getEdge().z, 'x');
    v2 = plane.solve(model->getEdge().x, model->getEdge().z, 'y');
    v3 = plane.solve(model->getEdge().x, 0.0, 'z');
    _v1 = plane.solve(model->getEdge().y, 0.0, 'x');
    _v2 = plane.solve(0.0, 0.0, 'y');
    _v3 = plane.solve(0.0, model->getEdge().y, 'z');

    if (math::calculateTriangleArea(v1, v2, v3)
        > math::calculateTriangleArea(_v1, _v2, _v3)) {
        v1 = _v1;
        v2 = _v2;
        v3 = _v3;
    }
} else
if ((a >= 0.0 && b < 0.0 && c < 0.0 && d < 0.0) ||
    (a < 0.0 && b >= 0.0 && c >= 0.0 && d >= 0.0)) {
    v1 = plane.solve(model->getEdge().y, model->getEdge().z, 'x');
    v2 = plane.solve(0.0, model->getEdge().z, 'y');
    v3 = plane.solve(0.0, model->getEdge().y, 'z');
    _v1 = plane.solve(0.0, 0.0, 'x');
    _v2 = plane.solve(model->getEdge().x, 0.0, 'y');
    _v3 = plane.solve(model->getEdge().x, 0.0, 'z');

    if (math::calculateTriangleArea(v1, v2, v3)
        > math::calculateTriangleArea(_v1, _v2, _v3)) {
        v1 = _v1;
        v2 = _v2;
        v3 = _v3;
    }
```

```
    } else {
        throw std::logic_error("Plane doesn't intersect model volume.");
    }
}

point_vec SlicerLSWR::rasterize_XY_XZ(const Point<int>& start,
                                      const Point<int>& masterEnd,
                                      const Point<int>& baseEnd) const {
    point_vec points;

    int modelEdgeX = model->getEdge().x, modelLength = model->getLength();
    int signX = (start.x > masterEnd.x) ? -1 : 1;
    int signY = (start.y > masterEnd.y) ? -1 : 1;
    int signZ = (start.z > baseEnd.z) ? -1 : 1;

    // Master line rasterization.

    int indexL, indexR, sizeC, iC, iL = 0, iR = 0;
    int sizeLR = (signX < 0) ?
        start.x - masterEnd.x + 1 :
        masterEnd.x - start.x + 1;

    if (signY < 0) {
        sizeC = start.y + 1;
        iC = start.y;
    } else {
        sizeC = masterEnd.y - start.y + 1;
        iC = 0;
    }

    int* C = new int[sizeC];
    int* L = new int[sizeLR];
    int* R = new int[sizeLR];

    int prevX = start.x;
    L[iL++] = start.y;

    // Bresenham's line algorithm.
        int deltaX = abs(masterEnd.x - start.x);
        int deltaY = abs(masterEnd.y - start.y);
        int x = start.x, x2 = masterEnd.x;
        int y = start.y, y2 = masterEnd.y;

        for (int error = 0; y != y2; y += signY, iC += signY) {
            if (prevX != x) {
                C[iC] = signX;
                L[iL++] = y;
                R[iR++] = y - signY;
                prevX = x;
            } else {
                C[iC] = 0;
            }

            error += deltaX;
            if (error * 2 >= deltaY) {
                x += signX;
                error -= deltaY;
            }
        }
```

```
    if (prevX != x) {
        C[iC] = signX;
        L[iL] = y;
        R[iR++] = y - signY;
    } else {
        C[iC] = 0;
    }
// !Bresenham's line algorithm.

R[iR] = masterEnd.y;

indexL = (signX < 0) ?
    ((start.x > modelEdgeX) ? start.x - modelEdgeX : 0) :
    ((start.x < 0) ? -start.x : 0);
indexR = sizeLR - 1;

// Base line rasterization.

prevX = start.x;

// Bresenham's line algorithm.
    int deltaZ = abs(baseEnd.z - start.z);
    int z = start.z, z2 = baseEnd.z;
    deltaX = abs(baseEnd.x - start.x);
    x = start.x, x2 = baseEnd.x;

    if (deltaZ < deltaX) {
        for (int error = 0; x != x2; x += signX) {
            int x0 = (signX < 0) ?
                ((x > modelEdgeX) ? modelLength : x) :
                ((x < 0) ? -1 : x);

            if (prevX != x) {
                if (indexL != 0) --indexL;
                if (signX < 0) {
                    if (x + 1 < sizeLR) --indexR;
                } else {
                    if (modelLength - x < sizeLR) --indexR;
                }
                prevX = x;
            }

            int limitL = L[indexL], limitR = R[indexR];

            for (y = limitL; y != limitR; y += signY) {
                x0 += C[y];
                points.push_back(Point<int>(x0, y, z));
            }
            x0 += C[y];
            points.push_back(Point<int>(x0, y, z));

            error += deltaZ;
            if (error * 2 >= deltaX) {
                z += signZ;
                error -= deltaX;
            }
        }
    } else {
```

```
            for (int error = 0; z != z2; z += signZ) {
                int x0 = (signX < 0) ?
                    ((x > modelEdgeX) ? modelLength : x) :
                    ((x < 0) ? -1 : x);

                if (prevX != x) {
                    if (indexL != 0) --indexL;
                    if (signX < 0) {
                        if (x + 1 < sizeLR) --indexR;
                    } else {
                        if (modelLength - x < sizeLR) --indexR;
                    }
                    prevX = x;
                }

                int limitL = L[indexL], limitR = R[indexR];

                for (y = limitL; y != limitR; y += signY) {
                    x0 += C[y];
                    points.push_back(Point<int>(x0, y, z));
                }
                x0 += C[y];
                points.push_back(Point<int>(x0, y, z));

                error += deltaX;
                if (error * 2 >= deltaZ) {
                    x += signX;
                    error -= deltaZ;
                }
            }
        }

        int x0 = (signX < 0) ?
            ((x > modelEdgeX) ? modelLength : x) :
            ((x < 0) ? -1 : x);

        if (prevX != x) {
            if (indexL != 0) --indexL;
            if (signX < 0) {
                if (x + 1 < sizeLR) --indexR;
            } else {
                if (modelLength - x < sizeLR) --indexR;
            }
        }

        int limitL = L[indexL], limitR = R[indexR];

        for (y = limitL; y != limitR; y += signY) {
            x0 += C[y];
            points.push_back(Point<int>(x0, y, z));
        }
        x0 += C[y];
        points.push_back(Point<int>(x0, y, z));
    // !Bresenham's line algorithm.

    delete[] C;
    delete[] L;
    delete[] R;

    return points;
}
```

Додаток 5

Копія презентації